**UNIVERSIDAD AUTÓNOMA DE MADRID**

**ESCUELA POLITÉCNICA SUPERIOR**




**Grado en Ingeniería Informática**

# TRABAJO FIN DE GRADO

## Diseño de una herramienta para la gestión de pistas deportivas


**Jorge Yubero Cabronero**
**Tutor: David Arroyo Guardeño**


**Mayo 2016**

# Diseño de una herramienta para la gestión de pistas deportivas


## Autor: Jorge Yubero Cabronero

**TUTOR: David Arroyo Guardeño**

**Departamento de Ingeniería Informática**
**Escuela Politécnica Superior**
**Universidad Autónoma de Madrid**


**Mayo de 2016**

# Resumen

De un tiempo a esta parte el interés y la posibilidad de las personas de acceder a todo tipo de información desde cualquier lugar y en cualquier momento haciendo uso únicamente de un dispositivo móvil, ha crecido exponencialmente Con tantas facilidades, y sumado a la creciente venta de dichos dispositivos, la sociedad ha empezado a acomodarse y a requerir que el acceso a la información y los datos sea fiable, seguro e instantáneo

Es por esto mismo que el desarrollo de aplicaciones para dispositivos móviles lleva unos años en constante crecimiento. Debido a este crecimiento, existe una corriente de migración de webs obsoletas a aplicaciones móviles para renovar su interfaz de cara al usuario y recuperar la usabilidad de la misma. Este proyecto de fin de grado, viene dado a causa de esta corriente migratoria.

Desde el momento de su creación la **Liga Cooperativa de Baloncesto de Madrid** disponía de una web que con el paso de los tiempos se fue quedando obsoleta, además, que no era adaptable a su visualización en los navegadores web de los dispositivos móviles. Por esta razón, para facilitar la tarea de gestión a los administradores de la liga, y, sobre todo, para proporcionar al usuario una herramienta con la que estar al día sobre todas las novedades de la liga, y poder interactuar con ella, se decido la migración de la web a una aplicación para dispositivos móviles, con sistema operativo Android actualmente.

En este trabajo de fin de grado se recoge el proceso seguido para la creación de una aplicación Android que consuma de un servicio web *RESTful PHP* y de una base de datos *MySQL*. Las conversaciones entre el cliente Android y el servicio web se llevarán a cabo en formato *JSON*.

Esta herramienta proporciona al usuario la posibilidad de autenticarse como jugador de la liga y acceder a cierta información a la que solo pueden acceder los jugadores, puede consultar la clasificación de su equipo, así como los partidos previstos para el día de hoy y posteriores en un calendario. Puede, además, consultar el estado y la dirección de las canchas más utilizadas, su parada de metro y autobús más cercana, ver su horario, etc.

Incluso, si dicho usuario, resulta ser, además de jugador, el administrador de su equipo, podrá acceder a otra serie de funciones, que, si no fuese administrador, no podría consultar. Dichas funciones son las correspondientes a fijar un partido contra un equipo rival del mismo grupo y modificar el resultado de un partido que previamente haya fijado.

# Palabras clave

Android, PHP, JSON, MySQL, Liga Cooperativa de Baloncesto de Madrid, dispositivos móviles

# Abstract

Nowadays, mobile phones enable users to access their data just by using their network connection. Indeed, people use mobiles in everyday life, such as consulting the news, checking out the weather, and other activities that previously were performed using other technological media. Moreover, mobile access to all these services is achieved in a faster and more reliable way than a few years ago.

Consequently, the development of mobile applications is very important in the current technological scenario. This trend has also motivated the upgrade of outdated websites in order to provide a proper access to mobile devices.

This final bachelor's project is intended to carry out the upgrade of the website of the **Liga Cooperativa de Baloncesto de Madrid**. The old website allows users to manage their basketball teams, checking out the league table, the calendar, etc. Currently, this website is not accessible on in modern mobile devices. This is the reason why the **Liga** needs a mobile application to manage all the functionalities required by the members. In addition, this app should be able to show information about the basketball courts and the nearest bus and *metro* stations.

Along this document, the analysis, design and validation of this new website are discussed. In specific, it includes the main aspects related to the development of a PHP web service, a JSON protocol, and an Android application. Therefore, the main result of this bachelor's project endow the players of the Liga Cooperativa de Baloncesto de Madrid with a means to organize the shared resources (through an administrator role) and to consult the information regarding matches, the league table, etc.



# Keywords

Android, PHP, JSON, MySQL, Liga Cooperativa de Baloncesto de Madrid, matches, Games, Mobile Phones,

# Agradecimientos

*Quiero dedicar y agradecer este trabajo a todas aquellas personas que de manera directa o indirecta me han servido de inspiración y ayuda a la hora de realizarlo. A su vez, agradecer también el apoyo recibido de todas las personas que han estado junto a mi a lo largo de toda la carrera.*

*En primer lugar, quiero agradecer a mi tutor, David, todo la ayuda prestada y los consejos que han servido para mejorar notablemente el proyecto.*

*También quiero agradecer su apoyo a todos los compañeros que he tenido a lo largo de la carrera, los que conocí desde el primer día y los que se han ido incorporando a lo largo de la carrera a mi circulo. Especial mención requieren Miguel, Luis, Óscar y Yoel por su inestimable apoyo en momentos difíciles y su gran actitud en los mejores momentos.*

*Agradecer también a mi familia todo el esfuerzo realizado para que, a día de hoy, yo sea la persona que soy y este en el sitio en el que estoy, al cual no habría llegado sin ellos. Especial mención a los que a día de hoy me acompañan, y también a los que ya no están.*

*Gracias papa, mama y Andrés por soportarme y ayudarme a mejorar. No sería lo que soy sin vosotros.*

*Jorge Yubero Cabronero*

*#StriveForGreatness*

# ÍNDICE DE CONTENIDOS

# ÍNDICE DE IMÁGENES

# ÍNDICE DE TABLAS DE REQUISITOS



# ÍNDICE DE TABLAS DE LA BASE DE DATOS



# ÍNDICE DE MAQUETAS DE LA FASE DE DISEÑO

# 1. Introducción

## 1.1 Motivación

Hace casi 4 años ya, a finales del verano de 2012, surgió una iniciativa vecinal en los barrios de Hortaleza y Chamartín de la ciudad de Madrid. Gente con una misma pasión se juntó para crear la primera **Liga Cooperativa de Baloncesto de Madrid**. Esta liga pretendía alejarse de los estándares de las ligas clásicas de baloncesto existentes en Madrid: las ligas de la federación y las ligas municipales.


Imagen 1.- Logo de la Liga Cooperativa de Baloncesto Temporada 2015/2016. Fuente: www.ligacooperativabasket.org

La comisión que creó la liga, pretendía crear una liga gratuita, abierta a todo aquel que quiera apuntarse, cooperativa, igualitaria, autogestionada y horizontal y asamblearia.

Se trata de una liga **gratuita** frente a las dos modelos de liga mencionadas anteriormente y de las que este proyecto de liga trataba de distanciarse. Es una liga en la que no hay que pagar nada, ni por inscribirse, ni por jugar, ni por participar. Esto es posible gracias a la organización de los equipos, los cuales han de ponerse de acuerdo en una fecha y un lugar para disputar dicho partido, eligiéndose prioritariamente las canchas públicas que existen en ambos barrios. Es una liga **abierta**, porque todo aquel que quiera apuntarse, puede hacerlo. En esta liga solo existen dos requisitos de participación. El primero es tener ganas de jugar al baloncesto, y el segundo es que tienen que primar los valores de la deportividad por encima de la victoria o derrota. Otra de las características de esta liga es, que, al contrario de las otras dos ligas, es que se confía en la deportividad del jugador, por encima del ansia de victoria que pueda existir, y es por esto, que en esta liga no existe la figura del árbitro. Cada equipo pita sus faltas, y se trata de buscar un acuerdo que satisfaga a ambos equipos en caso de que existan desencuentros durante la disputa del partido.

Se dice que es **cooperativa**, porque pese a tratarse de una liga gratuita, existen gestiones que se tienen que realizar, y se quiere que se repartan entre todos los participantes, para fomentar la cooperación entre los propios jugadores. Igualitaria porque no se rechaza la entrada y participación a nadie por ninguna situación, ya sea la edad, el sexo, la religión, e incluso, la calidad baloncestística. No existe límite, ni superior, ni inferior por el cual alguien pueda quedar excluido.

Y sobre el resto de cosas, es una liga horizontal y asamblearia, debido a que todas y cada una de las decisiones que afectan a la liga se toman en una asamblea a la que puede apuntarse cualquier persona. Estas asambleas son obligatorias, para al menos, un jugador de cada equipo, para mantener la representatividad de toda la liga, y que no se tomen decisiones al margen de ningún equipo. Se organizan tres asambleas anuales que son al principio de la liga, para organizar la nueva temporada, controlar la inscripción de equipos, y dar la bienvenida a los nuevos equipos que se apunten esa temporada, así como para explicarles el funcionamiento de la liga. Otra asamblea se realiza terminada la primera fase

de la liga (sobre finales de enero o principios de febrero) en la que se analiza el trabajo realizado por las distintas comisiones que forman la liga, y para organizar la segunda fase. Y, otra asamblea a finales de abril, para organizar la fase final de la liga, y comentar y resolver los posibles problemas que hayan surgido a lo largo de la temporada. En esta última asamblea se empiezan a detallar los detalles de las finales de la liga, como pueden ser la fecha, el lugar, la hora, etc.

Como cada equipo se encarga de organizar sus partidos (siempre de acuerdo con el rival), se hacía necesaria la existencia de una herramienta informática que permitiese ir actualizando los resultados, pero, sobre todo, donde ir apuntando los partidos que se van fijando para evitar que se organicen dos partidos sobre una misma pista. Con esta idea, un grupo de miembros de la liga formó la comisión web de esta, y crearon una web simple donde ir realizando estas acciones. Con el paso del tiempo, y el auge de las tecnologías móviles, la web empezó a quedarse desfasada y el acceso desde dispositivos móviles resultaba muy difícil y costoso.

Por esta razón, se decidió crear una aplicación móvil que facilitase esta tarea a los usuarios de la liga. Además de guardar los resultados y fijar los partidos, se decidió que la aplicación realizase otras tareas.

Los objetivos del presente proyecto son, precisamente la creación de una nueva base de datos actualizada de la liga, así como una capa web mediante un servicio web *RESTful* en PHP que facilite la tarea a los usuarios de la aplicación

## 1.2 Objetivos

El principal cometido de este proyecto es proporcionar a la Liga Cooperativa de Madrid una herramienta que permita agilizar las gestiones de los jugadores y/o equipos con la propia liga, además de facilitar un nuevo medio de gestión de ésta.

Puesto que anteriormente solo existía una web a la que no se podía acceder a través de un dispositivo móvil y una hoja Excel desactualizada donde los equipos anotaban sus resultados. Con miras a que fuera posible poder llevar un control real y exhaustivo por parte de los coordinadores, se hacía necesario un nuevo método para interactuar entre la liga y los equipos, y entre los propios equipos. Es por esto que se consideró conveniente la realización de un servicio web *RESTful* en PHP para poder ser consumido tanto desde la web existente, como desde una aplicación móvil. En esta memoria se describe todo el proceso asociado a la concreción tanto del servicio web y la base de datos [1], como de la aplicación móvil Android. Este proceso de concreción incluye todas las funcionalidades demandadas en primera instancia por los usuarios de la liga, pero también se analizaron funcionalidades adicionales. Es más, el proceso de análisis, diseño e implementación se efectúo de formar que el resultado final pudiera ser exportable a otros proyectos o contextos de aplicación.

La elección de Android como cliente móvil está motivada por la existencia de al menos un usuario con acceso a un dispositivo móvil con sistema operativo Android, por lo que se prima la usabilidad de la solución propuesta por encima de otros factores.

Al tratarse de una liga autogestionada y que permite una gran flexibilidad con la disputa de los partidos, simplemente han de ponerse de acuerdo los dos equipos para jugar, y al

creciente número de equipos inscritos, era esencial, disponer de una herramienta, con un algoritmo, que calcule cuales son las canchas libres dado un día y una hora, para evitar problemas del estilo a programar dos partidos el mismo día, a la misma hora y en el mismo lugar.
Con esto se concluye que existe la necesidad de una herramienta para que los usuarios puedan interaccionar con la liga de una manera mucho más dinámica, ágil y sencilla, pudiendo gestionar sus partidos, comprobar la clasificación y el calendario o realizando acciones como consultar el estado y dirección de las canchas disponibles.

## 1.3 Organización de la memoria

La memoria de este proyecto está organizada según los pasos que sigue un proyecto a lo largo de su ciclo de vida. Estos pasos son análisis, diseño, desarrollo, pruebas, integración, pruebas de integración y entrega al cliente.

En la sección de Introducción se desarrolla la motivación y los objetivos de este trabajo, la organización de la memoria, así como una breve introducción del tema correspondiente a la memoria. En la sección de Estado del arte se analizará el estado de las tecnologías usadas (Android, PHP y SQL), así como por qué se ha elegido su utilización sobre otras tecnologías que podrían producir resultados similares. En el apartado de Metodologías y patrones empleados se hablará acerca de las metodologías empleadas a la hora de afrontar el proyecto, así como los patrones utilizados en su diseño. Se desarrollará una breve explicación sobre cada uno de ellos, y luego se relacionará con la parte que le corresponda del proyecto, además, como es un tema extenso, se ha añadido información adicional en los anexos finales de esta memoria. En la sección de Análisis se comentaran los distintos procesos de análisis de cada *sprint* a través de sus requisitos. En el capítulo sobre Diseño e implementación se hablará sobre el diseño de un proyecto, y centrándonos en el proyecto actual, sobre el diseño de una aplicación Android con web PHP y base de datos SQL [1]. También se mostrarán las maquetas de la aplicación. En el apartado de Desarrollo se comentarán todos los detalles acerca del desarrollo de todas y cada una de los *sprints* en los que se ha dividido el proyecto. En la sección de Integración, pruebas y resultados se desarrollará todo el tema referente a las pruebas y la integración de los *sprints* tras la finalización de cada uno de ellos a lo largo del ciclo de desarrollo del proyecto. En el capítulo sobre Conclusiones y trabajo futuro se desarrollaran las conclusiones sobre el proyecto, una posible mejora para el proyecto y una opinión personal acerca del trabajo realizado y los aprendizajes adquiridos.

# 2. Estado del arte

Antes de elegir las tecnologías que serán utilizadas en el desarrollo del proyecto, se estudiaron todas aquellas que podrían permitir realizar la aplicación deseada. Después de analizarlas, se ha hecho un estudio para ver qué tecnología es la más adecuada para desarrollar este proyecto.

Como la aplicación debe tener tres capas, se tienen que realizar tres análisis de las distintas tecnologías: primero, una tecnología para dispositivos móviles, una segunda tecnología para el desarrollo web, y tercero una tecnología para bases de datos.

En los apartados que siguen a este párrafo se detallaran todas las tecnologías seleccionadas y se expondrán las razones y motivos que han llevado a elegir la tecnología que finalmente se use durante el proyecto.

## 2.1 Dispositivos móviles

En este primer apartado, la memoria se centrará en las tecnologías para dispositivos móviles.

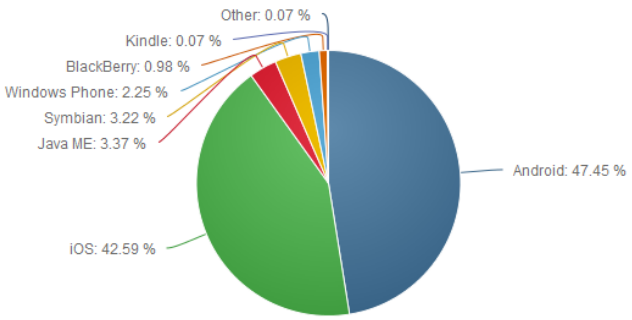



Imagen 2.- Reparto de los SO de dispositivos móviles en 2015. Fuente: https://blog.uchceu.es/informatica/ranking-de-sistemas-operativos-mas-usados-para-2015/

Como se puede observar en el gráfico de Imagen 2, los dos sistemas operativos más utilizados en los dispositivos móviles, en el año 2015, y con una gran diferencia sobre el resto son Android con un 47.45%, el sistema operativo de Google, y el sistema operativo de *Apple Inc*. *iOS*. Por tanto, estos dos sistemas operativos serán los que se tengan en cuenta durante el análisis de las tecnologías para dispositivos móviles por ser los dos sistemas operativos más usados en el mercado de dispositivos móviles

**Android** es un sistema operativo ideado para dispositivos móviles con pantalla táctil, que está basado en el núcleo Linux. En sus inicios, Android fue desarrollado por la empresa Android Inc., que fue respaldada económicamente por Google y posteriormente comprada por esta. Su historia comenzó oficialmente el 5 de noviembre de 2007 cuando se crea la ***Open Handset Alliance***, y además se presenta la primera versión del sistema operativo. Se llamaría Apple Pie y seria la versión 1.0 (**Android 1.0** ***Apple Pie*** como nombre completo). Ese mismo día, Google liberó la mayor parte del código Android bajo la licencia **Apache**, una licencia **libre** y de **código abierto**. A día de hoy, la última versión presentada y descargable es **Android 6.0** ***Marshmallow*****.**


Imagen 3.- Logo Android 6.0 Marshmallow. Fuente: http://chynsa.blogspot.com.es

La estructura del sistema operativo Android está basada en aplicaciones ejecutándose en un *framework* Java en una máquina virtual *Dalvik*, cuya compilación se realiza en tiempo de ejecución.

***iOS*** es el sistema operativo para dispositivos móviles de la empresa *Apple Inc*. Originalmente fue diseñado para teléfonos móviles (*iPhone*), pero al igual que Android, a día de hoy, está disponible para infinidad de dispositivos tecnológicos, como pueden ser reproductores de música (*iPod*), tabletas (*iPad*), e incluso relojes (*iWatch*). Este sistema operativo está diseñado de tal manera que no puede ejecutarse en *hardware* de terceros.

La primera versión, beta, del *iPhone SDK* apareció el 6 de marzo de 2008. Anteriormente a este sistema, el sistema que ejecutaban los *iPhone* (únicos miembros de la familia *iOS* en aquel momento) era el OS X. Se trataba de una versión modificada de un sistema conocido como *NewtonOS*. El nombre que recibió esta primera versión fue ***iPhone OS***, cuyo lanzamiento tuvo lugar el 29 de junio de 2007. A día de hoy, la última versión disponible es ***iOS* 9.3.2.**

Imagen 4.- Logo iOS 9. Fuente: http://www.vcpost.com/articles/92063/20150916/apple-ios-9-officially-released-early-download-attempts-failed.htm

Para el desarrollo de la aplicación se ha elegido usar el sistema operativo ***Android*** debido a razones tales como poseer una licencia gratuita y de código abierto, con lo que conlleva que cualquier persona, sin necesidad, de pagar un solo euro, puede realizar aplicaciones Android y disponer a su alcance de todas las herramientas y tecnologías que el sistema ofrezca a usuarios y desarrolladores. A pesar de esto, la publicación de aplicaciones en la tienda oficial de Android no es gratuita. Es necesaria la compra de una licencia de publicación ilimitada e infinita por 25$, mientras que la licencia de publicación de aplicaciones en la tienda oficial de iOS oscila en 99$ y 299$ al año.

Sin embargo, *iOS*, en el apartado de la optimización del código, es muy superior a Android, pues sus aplicaciones solo se pueden utilizar sobre *hardware* creado por la empresa, por lo que el código está especialmente optimizado para dicho *hardware*. Sin embargo, como Android se puede ejecutar sobre cualquier dispositivo (siempre que cumpla los requisitos mínimos de ejecución de la versión del SO), no se puede optimizar las instrucciones sobre un *hardware* predeterminado, pues esto perjudicaría a todos los dispositivos que no posean ese *hardware*.

Otra razón, y probablemente la razón de mayor peso a la hora de elegir el sistema operativo Android ha sido, que realizando un estudio sobre los futuros clientes de la aplicación (los equipos de la liga) se ha llegado a la conclusión de que en todos los equipos existe al menos un usuario con un teléfono Android que cumpla los requisitos de la aplicación (versión del sistema operativo mayor o igual que *Android* 5.0 *Lollipop*), mientras que no todos los equipos poseían algún usuario con acceso a un dispositivo que pudiese ejecutar el sistema operativo *iOS* 9, es decir, un *iPhone* o un *iPad*, por lo que predomina la usabilidad de la aplicación sobre la eficiencia de la misma, pues como queda demostrado en las aclaraciones anteriores, las aplicaciones de *iOS* son más eficientes.

Otro motivo más para la elección de Android sobre *iOS*, y esto es de cara al futuro, es la facilidad de integración de una aplicación con *APIs* de terceros. Debido a que algunas de las APIs más importantes de las que pueden ser utilizadas en la aplicación o solicitadas por los usuarios para futuras versiones, como puede ser la *API* de *Google Maps* o incluso Google Fotos. La inclusión de estas y otras *APIs* no tanto de *Google* sino de cualquier otro

desarrollador es mucho más sencilla en una plataforma Android que sobre cualquier plataforma *iOS*.

## 2.2 Web

Para esta capa se necesita una tecnología que funcione del lado del servidor, sea compatible con Android, o al menos, permita la interconexión entre el servidor y el dispositivo, y, además, permita acceder a una base de datos. Se seleccionan PHP y ASP.NET.

***PHP*** [5] es un lenguaje de uso general, de lado del servidor y que originalmente fue pensado para desarrollar webs con contenido dinámico. Fue el primer lenguaje que permitía codificarse en la misma página *HTML*. El código tiene que ser interpretado por un servidor web que posea un módulo que pueda procesar *PHP*. Tiene la ventaja de poder ser utilizado en casi todos los sistemas operativos y plataformas sin ningún tipo de gasto.

Fue creado en 1995 y su creador fue *Rasmus Lerdof* y lanzado bajo el nombre de *Personal Home Page Tools* o ***PHP Tools*** el 8 de junio del mismo año. Actualmente, la última versión disponible y con soporte por parte de los desarrolladores es la versión 7.0, lanzada el 3 de noviembre de 2015.

Imagen 5.- Logo de PHP. Fuente: http://php.net/manual/es/imagick.examples-1.php

Funciona bajo una licencia *PHP* y pertenece al grupo de software libre. La licencia *PHP* se trata de una licencia de software libre no *copyleft*. Con esta licencia se permite la redistribución del contenido siempre que se cumplan tres requisitos, que son la inclusión de la declaración de los derechos de autor de la licencia *PHP*, que la palabra *PHP* no se use en el título de las obras derivadas y que se incluya el siguiente anuncio bajo cualquier forma en la que se redistribuya el código:

> *This product includes PHP software, freely available from <http://www.php.net/software />*

***ASP.NET*** es un *framework* para aplicaciones web desarrollado y comercializado por Microsoft. Es utilizado para el desarrollo de sitios web dinámicos, aplicaciones web y servicios web XML.

Imagen 6.- Logo de Microsoft ASP.NET. Fuente: https://ingcamilorodriguez.wordpress.com/2012/05/10/modelos-de-programacion-en-asp-net-web-forms-mvc-y-web-pages/

La primera versión del *framework* apareció en enero del año 2002 siendo la tecnología que sucedería a *Active Server Pages* (ASP). Esta construido sobre la base del Common Language Runtime, lo que posibilita que los desarrolladores escriban código para *ASP.NET* utilizando cualquier lenguaje admitido por el *.NET Framework*.

Finalmente, la tecnología escogida ha sido ***PHP*** entre otras razones, por sus costes, **cero**, ya que es totalmente gratis y multiplataforma, por lo que se puede desarrollar en cualquier sistema operativo, y se hospeda en servidores Linux, mientras que *ASP.NET* al tratarse de un producto de Microsoft requiere algunos gastos, como puede ser, el propio sistema operativo *Windows*, pues solamente se puede desarrollar sobre él, la licencia de uso del entorno de desarrollo,

pues a pesar de existir una versión gratuita, no es todo lo completa que se necesitaría para un desarrollo a nivel profesional, y la licencia para utilizar el *Visual Studio Professional* 2015 cuesta 499$.

Otro motivo es el soporte, y a pesar de *ASP.NET* es propiedad de *Microsoft* y ofrece un gran soporte, no posee una comunidad de desarrolladores detrás, como si es el caso del lenguaje *PHP*.

Y, sobre todo, uno de los grandes motivos, a la hora de elegir *PHP* ha sido la facilidad de integración con *Android* y con la base de datos utilizada (sobre la que se hablara en el próximo capítulo), además de la facilidad de añadirle una capa de seguridad *SSL* y desplegarlo en un servidor *Apache*.

## 2.3 Base de datos

Para esta capa, se selecciona bases de datos que permitan la integración con PHP, que es la tecnología web elegida, y por tanto el modelo de nuestro patrón *MVC*. Las tecnologías seleccionadas son *PostGreSQL* y *MySQL*.


Imagen 7.- Logo de PostGreSQL. Fuente: https://platzi.com/blog/que-es-postgresql/

***PostGreSQL*** es un sistema de gestión de base de datos objeto-relacional basado en el proyecto POSTGRES. Se trata de una derivación libre, según el código *OpenSource*, de este proyecto y utiliza el lenguaje SQL92/SQL99.

Soporta la gran mayoría de las transacciones *SQL* y tiene a su disposición lenguajes como *Java* (*Android*) y *PHP* (capa web). Su primera versión operativa fue liberada en junio de 1989 a unos pocos usuarios. El nombre de *PostGreSQL* llega con la versión 6.0 en el año 1996.


Imagen 8.- Logo de MySQL. Fuente: http://www.ochobitshacenunbyte.com/2015/02/02/crear-una-base-de-datos-mysql-en-gnu-linux/

***MySQL*** es un sistema de gestión de bases de datos relacional con licencia bajo la *GPL* de la *GNU*. Fue desarrollado por la empresa sueca *MySQL* AB, que mantiene el *copyright* del código fuente del servidor SQL A pesar de tratarse de un sistema de código libre, la empresa ofrece una licencia comercial en la que ofrece soporte técnico al usuario.

*MySQL* surgió como un intento de conectar el gestor *mSQL* a las tablas propias de *MySQL AB*, usando sus propias rutinas a bajo nivel, lo que no resulto posible debido a la poca flexibilidad de *mSQL*. Debido a esto, tuvieron que desarrollar nuevas funciones que resulto proporcionando una interfaz SQL para sus tablas.

En sus inicios, *PostGreSQL* era más rico en funcionalidad, y *MySQL* era el más rápido de ambos, pero a día de hoy, las diferencias son mucho más sutiles, e incluso ínfimas, por lo que el proyecto podría haber sido realizado con cualquiera de los dos, debido a las grandes similitudes que tienen, mas allá, de las sutiles diferencias que los separan. Por lo que la elección de ***MySQL*** sobre *PostGreSQL* no reside en ninguna razón considerable.

# 3. Metodologías y patrones empleados

En este apartado de la memoria, se explicarán las metodologías y los patrones de diseño empleados en el desarrollo de la aplicación. En caso de una necesidad de información más amplia, el contenido de cada uno de los apartados siguientes, se desarrollará en los anexos.

## 3.1 Desarrollo agile

Durante años, los proyectos de desarrollo de software han seguido un ciclo de cascada o escalera, un desarrollo que para la mayoría de proyectos no es adecuado, pues esperar al final del mismo para evaluar su impacto puede hacer que realizar cambios sea costoso temporal y económicamente, pues si hay que rediseñar la aplicación, puede darse el caso de tener que empezar de cero.

No se tardó mucho en desarrollar una metodología que aceptase que los problemas no están bien definidos y se admitiese que el desarrollo del proyecto pueda ser dinámico y abierto a cambios. A esta metodología se la denomina **desarrollo *Agile*** [7], que se define como un marco de trabajo en el cual, por encima de todo, está la responsabilidad de cada individuo dentro del equipo durante el desarrollo, la existencia de una actitud de respuesta positiva frente a posibles cambios y al valor del WIP (*work in progress* o trabajo en progreso). Todo esto se encuentra dentro de un manifiesto publicado en el año 2001, cuyos doce principios se encuentran en .


Imagen 9.- Esquema del ciclo de vida de la *metodología agile*. Fuente: http://multipointit.com/mps/show_pages/22

En la Imagen 9 se puede observar el que sería el clásico esquema de un ciclo de vida de un proyecto de software aplicando la *metodología agile*. Se puede observar que en el inicio

del proyecto (el punto gris oscuro abajo a la derecha), el primer paso es definir los requisitos. Acto seguido se realiza una planificación a alto nivel, y se comienza después con el desarrollo y la implementación seguido de las correspondientes pruebas. Una vez acabadas todas las subtareas se le entregan al cliente para pruebas. El cliente que realiza las pruebas no suele ser el usuario final, y si este aprueba el funcionamiento correcto de la tarea o *sprint*, se prueba en el sistema final y se sube a producción. El proceso, resumido, correspondería a las fases de desarrollo (las partes de desarrollo e implementación y pruebas. Se suele realizar sobre un entorno muy cambiante), las fases de preproducción (aquellas partes en las que el cliente prueba la aplicación) y las fases de producción (aquella parte en la que, tras probar el entorno final, se considera la tarea como correcta). En caso de que el cliente considere que la aplicación no está correcta, este entregara los fallos/*bugs* encontrados que se incorporaran a la nueva iteración como nuevas partes a desarrollar e implementar. En caso de que el cliente si apruebe la tarea, y esta no sea la última, la parte de evaluar los errores/*bugs* correspondería a la parte de introducir la nueva tarea en la rutina del ciclo de la metodología

## 3.2 Método Scrum

De este manifiesto de 2001 nació el método de trabajo ***Scrum***, el cual establece que un proyecto se planifica dividiéndolo en varios bloques temporales llamados *sprints* o tareas, las cuales son divisiones más pequeñas y funcionales del proyecto que pueden ser entregados al cliente y pueden ser utilizados completamente por este último.

Estas tareas, pueden tener o no relación con la anterior, pero la base principal del método *scrum* es que, en cada tarea, el equipo evoluciona el proyecto de manera incremental basándose en los resultados completados de las tareas ya desarrolladas anteriormente.

Una de las claves más importantes del método *scrum* para subdividir el proyecto en tareas es limitar estas al espacio temporal. Lo recomendable, según el manifiesto anterior, es establecer objetivos a corto plazo, de no más de dos semanas de duración. Al hacer estas tareas tan cortas nos da un *feedback* mucho más rápido sobre los posibles cambios, así como que aumenta el poder creativo de cara a las siguientes tareas.

Otro de los fundamentos de la filosofía *agile* es que proporciona autonomía y responsabilidad a partes iguales entre todos los individuos que conforman el equipo, lo que implica involucrar en el proyecto a clientes, interlocutores, diseñadores, programadores, etc. Esto requiere la existencia de actitudes respetuosas y relaciones de confianza entre todos los miembros. Para ello, el método *Scrum* propone que, al inicio del día, se lleve a cabo una reunión o *daily* de 15 minutos de duración aproximadamente en la que cada miembro expone el trabajo realizado desde la *daily* anterior y cuáles serán los pasos a seguir durante la jornada actual, siendo el objetivo fomentar nuevos planteamientos y garantizar el correcto avance de cada tarea en colaboración con cada uno de los miembros.

Se puede observar una explicación detallada de un sprint en un proyecto en el anexo **A**.

## 3.3 Método Kanban

***Scrum*** define varios aspectos de la metodología para organizar el trabajo, como, por ejemplo, disponer de un ***scrum master*** (jefe de proyecto, normalmente) que dinamice y facilite la resolución de las tareas, y la existencia de una ***scrum board***, donde el proceso quede visualizado y permita la auto-organización del equipo y las tareas.

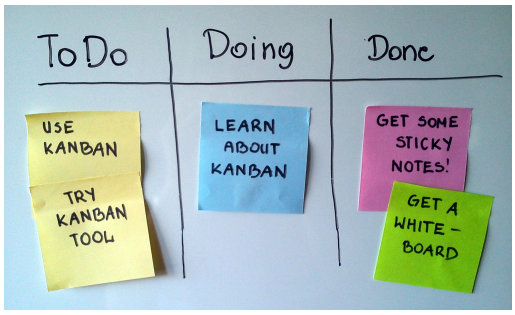



Imagen 10.- Pizarra con post-it resumiendo la metodología Kanban. Fuente: http://fercontreras.com/kanban/

En este apartado entra el denominado **método *Kanban***, cuyo nombre viene del japonés y significa **tarjetas visuales** (kan = visual y ban = tarjeta). Este método fue ideado por Toyota y se apoya en tres principios o reglas básicas: visualizar el trabajo y el flujo de este, determinar el límite de trabajo en curso y medir el tiempo en realizar una tarea

### 3.3.1 Visualizar el trabajo y flujo de este

La idea con la que *Toyota* concebía *Kanban* era la de tener un muro o un tablero, donde todos los integrantes de cada grupo pudiesen ver todas las tareas existentes asignadas al proyecto, ver en qué fase están (desarrollo, pruebas, recamara, etc.) y ver al miembro al que estaban asignadas las tareas. Los comienzos del método *Kanban* se llevaron a cabo sobre pizarras con post-it como se puede observar en la Imagen 10**.**

Con el incremento de uso del método *Kanban* se han ido creando nuevas columnas "estándares" así como herramientas informáticas que ayudan en este proceso. Una de estas herramientas, y la utilizada en este proyecto, es la herramienta ***Kanbanchi***. Se trata de una herramienta gratuita con ciertas funcionalidades de pago, más orientada a proyectos con más usuarios y de mayor envergadura que el que nos atañe en esta memoria. Permite la creación o eliminación de tantas columnas del tablero como deseemos, para adaptarse a las necesidades del proyecto.

El tablero durante el proyecto ha sido explicado en el anexo **B**, además de los pasos de una tarjeta en el mismo.

### 3.3.2 Determinar el límite de "*trabajo en curso*"

Otra de las ideas principales en las que está basado el método ***Kanban*** es en mantener siempre el orden dentro de las tareas que se están llevando a cabo, en otras palabras, la idea de *Kanban* es centrarse en cerrar las tareas que se tienen abiertas en lugar de abrir otras nuevas. Es por esto que existe el límite de tareas en determinadas columnas del tablero.

Como es lógico, solamente existen límites de tareas dentro del grupo de tareas en desarrollo. Lo difícil de esta regla *Kanban* es limitar el trabajo en curso o *WIP* (de su traducción en ingles *work in progress*) para no producir un cuello de botella, o para evitar que haya demasiado margen para abrir nuevas tareas y no cerrar nunca las existentes en desarrollo.

### 3.3.3 Medir el tiempo en realizar una tarea

Esta regla o norma *Kanban*, está enfocada más de cara al cliente o al jefe de proyecto que a los propios integrantes del grupo. Con esta norma, lo que se pretende es crear un hábito de trabajo, de manera que pueda estimarse un coste temporal sin mucho error, de cara a dar una estimación al cliente. Además, también sirve para que el jefe del proyecto pueda evaluar el rendimiento del equipo para cada una de las tareas.

El tiempo se mide de manera distinta para el jefe del proyecto y para el cliente. Para el cliente se comienza a medir desde el momento en el que la tarea tiene una tarjeta en el tablero (sea cuál sea la columna en la que se cree) hasta que la tarjeta llega a la columna de producción.

Sin embargo, para el jefe de proyecto, la medición comienza en el momento en el que la tarjeta entra en cualquiera de las columnas del grupo en desarrollo por primera vez (aunque en la fase de verificación se detecten *bugs* y se envíe la tarjeta al *backlog* por necesidades del tablero, esta sigue contando) hasta que la tarjeta es colocada en la columna de en producción.

## 3.4 Modelo Vista Controlador (MVC)

El *MVC* de su nombre completo Modelo Vista Controlador, es un patrón de diseño muy utilizado, con el cual, se logra una completa abstracción entre las tres capas que pueden formar un proyecto de software. Dichas capas corresponden a cada una de las palabras del nombre del patrón de diseño.

La idea del *MVC* [6] es conseguir una abstracción total entre las capas implicadas. Con esto se facilita el desarrollo, pero sobre todo el futuro mantenimiento. Para ello se basa, sobre todo, en la reutilización de código y la separación de conceptos.

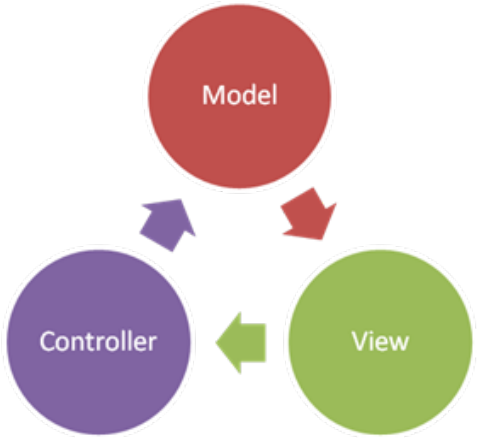



Imagen 11.- Diagrama básico del MVC.
Fuente:
http://www.codeproject.com/Tips/669195/MVC-Introduction

El **modelo** representa la capa de datos de la que se nutre la aplicación desarrollada. Es la capa encargada de gestionar todos los accesos a los datos. Esta capa, como se puede observar en Imagen 11, es la encargada de facilitar los datos requeridos a la vista, para que estos puedan ser mostrados al usuario a través de la interfaz gráfica.
Esta interfaz gráfica es controlada por la capa de la **vista**, que es la encargada de gestionar tanto dicha interfaz como las interacciones que el usuario mantiene con esta. Con cada interacción, el usuario ejecuta una acción sobre la vista que esta traslada al **controlador** que es el encargado de interpretar dicha acción y responder ante ella, normalmente contactando con el modelo para solicitar datos.
A su vez, el controlador, puede solicitar a la vista una nueva manera de mostrar los mismos datos anteriores, por lo que no necesitaría contactar con el modelo, véase el caso de un scroll en pantalla.

# 4. Análisis

En esta sección se tratará todo aquello relacionado con el primer paso en el desarrollo de un proyecto, según las reglas de la ingeniería del software, el **análisis.**

Se considera análisis de un proyecto de software a aquellas etapas que se ocurren desde el momento en el que un cliente encarga un proyecto hasta que este es aprobado por aceptado tanto por el equipo de desarrollo como por el cliente.

Estas etapas incluyen las reuniones con el cliente para aclarar toda la funcionalidad que debe llevar a cabo la aplicación, aquellos detalles no funcionales, pero que forman parte también del conjunto de requisitos que desea obtener el cliente.

También, forman parte de esta etapa, los estudios de dichos requisitos por parte de los analistas, así como un estudio a fondo de la viabilidad del proyecto.

Una vez terminadas todas estas etapas, el equipo debe elaborar un informe para el cliente en el que se debe reflejar la viabilidad del proyecto, así como todos y cada uno de los requisitos especificados por el cliente, divididos en las unidades atómicas más pequeñas posibles.

Sin embargo, cuando el proyecto realizado se hace bajo la *metodología agile* y *scrum*, el proceso de análisis varia respecto a lo especificado anteriormente. Varia, sobre todo en la forma de actuar, pues suele ser un trabajo más conjunto entre cliente y equipo de desarrollo. El cliente participa en las reuniones del equipo, expone sus ideas de cómo afrontar el problema, etc.

Otra de las variaciones es que el análisis no se ha realizado para toda la aplicación de manera conjunta, sino que se han ido analizando los distintos *sprints* de los que se compone el proyecto. Debido a esto, el apartado se compondrá de tantos apartados como *sprints* se han realizado, en los que se detallara el proceso analítico llevado a cabo para cada uno de ellos. Además, se expondrán los requisitos tanto funcionales como no funcionales de cada uno de ellos

Se consideran requisitos funcionales aquellos que recogen la funcionalidad que debe tener la aplicación, indicando qué cosas debe hacer, y se considera requisito no funcional a aquel requisito centrado en el rendimiento, a aquel que impone restricciones en el diseño o la implementación y a aquel que aporta cualidades que debe cumplir el producto final.

En referencia a los requisitos no funcionales, existen varios que son comunes a todos los *sprints* y que, por tanto, se verán reflejados en este mismo apartado.

### 4.1.1 Requisitos no funcionales generales

| Nombre | Requerimiento | Descripción |
|---|---|---|
| **RNF_General_1** | Fácil de usar | Se debe tratar de una aplicación que no necesite ningún tipo de tutorial ni explicación para poder ser usada desde el primer momento, contribuyendo a mejorar la experiencia del usuario dentro del ecosistema de la liga |
| **RNF_General_2** | Explicativa | En aquellos casos en los que el uso de la aplicación, no sea trivial, debe ser lo suficientemente explicativa como para ayudar al usuario |
| **RNF_General_3** | Acorde a los valores de la liga | Deben respetarse los valores de la liga, la cual es autogestionada, inclusiva, independiente. Por ello, la aplicación no debe llevar publicidad ni contener inversiones mediante micro pagos. Debe ser además una aplicación que incluya a todos dentro de ella, sin que nadie se sienta discriminado por ningún tipo de actividad. |
| **RNF_General_4** | Acorde a la imagen de la liga | Debe presentar una imagen acorde a la liga, debe predominar el color naranja representativo de la liga, debe aparecer su logo, los carteles de la galería deben representar a la liga proporcionando una buena imagen de ella. |
| **RNF_General_5** | Ágil | Debe ser una aplicación ágil y que responda rápido. Se eligió la idea de realizar una aplicación móvil para solventar los problemas que presenta la actual web en temas de rendimiento desde dispositivos móviles, por lo que la aplicación debe suplir a la web de la mejor manera posible. Se debe establecer un timeout de operaciones no mayor al minuto de duración. |
| **RNF_General_6** | Adaptable | La aplicación móvil, que es la única interfaz de usuario dentro del proyecto, debe ser capaz de adaptarse a distintos tamaños pantalla de teléfono, no depender de instrucciones de un procesador o un componente grafico especifico, pues como se explicó anteriormente, Android se desarrolla en todo tipo de terminales, muy diferentes unos de otros. |

Tabla de Requisitos 1.- Requisitos no funcionales generales de la **aplicación**

Si se considerase oportuno, en alguno de los *sprints* introducir algún nuevo requisito no funcional, este se explicará en los apartados correspondientes a cada uno de los *sprints*

## *4.2 Login [SPRINT 1]*

Para esta primera tarea, el cliente solicitó, una función de *login* para que aquellas personas registradas, puedan acceder a la aplicación, así como obtener el rol de dicho usuario (administrador o usuario). También se solicita, que todo el mundo pueda acceder, sin realizar el *login*, pero con un rol de visitante (simplemente puede ver ciertas pantallas, sin llegar a introducir ningún dato).

### 4.2.1 Requisitos funcionales

| Nombre | Requerimiento | Descripción |
|---|---|---|
| **RF_Sprint_1.1** | Comunicación a través de canal seguro | Se debe mantener la seguridad y la integridad de las comunicaciones, en especial durante el envío de datos críticos, como la contraseña. |
| **RF_Sprint_1.2** | Comprobación de credenciales | Se debe comprobar la existencia en base de datos de la tupla <nombre, contraseña> que coincida con la introducida por el usuario. |
| **RF_Sprint_1.3** | Almacenamiento de credenciales | Se debe permitir, bajo elección del usuario, almacenar las credenciales del usuario en un fichero de preferencias. |
| **RF_Sprint_1.4** | Obtención de un ID único por sesión | Se debe obtener un ID único por cada sesión iniciada y almacenarlo para mantener un control de los ID creados. |
| **RF_Sprint_1.5** | Obtención del rol de usuario | Se debe obtener de la base de datos la categoría de usuario a la que pertenece aquel que realiza el *login*. |
| **RF_Sprint_1.6** | Permitir entrada como visitante | Se debe permitir la entrada de usuarios no registrados bajo una categoría "Visitante" con acceso restringido a distintas áreas. |
| **RF_Sprint_1.7** | Cambio de contraseña | Se debe permitir realizar el cambio de contraseña tras la primera entrada a la aplicación. |

Tabla de Requisitos 2.- Requisitos funcionales del *sprint* ***Login***

### 4.2.2 Requisitos no funcionales

No se aporta ningún requisito extra a los ya mencionados anteriormente.

## *4.3 Consultar canchas vacías [SPRINT 2]*

Suponiendo una duración media de partido de aproximadamente una hora y cuarto (75 minutos), el cliente solicita un algoritmo, con el que, dada una fecha y una hora, se devuelvan las canchas con pistas libres.

### 4.3.1 Requisitos funcionales

| Nombre | Requerimiento | Descripción |
|---|---|---|
| **RF_Sprint_2.1** | Obtención del listado completo de canchas | El algoritmo tiene que ser capaz de obtener el listado completo de las canchas almacenadas en la base de datos para su posterior análisis. |
| **RF_Sprint_2.2** | Obtención del número de pistas por cancha | El algoritmo tiene que ser capaz de obtener el número de pistas que hay en cada cancha. Una cancha es el recinto, y las pistas son los distintos campos de baloncesto para jugar. |
| **RF_Sprint_2.3** | Obtención del número de partidos en una cancha durante el horario critico | El algoritmo tiene que poder obtener el número de partidos que hay ya fijados en cada una de las canchas dentro del horario crítico. Se define horario crítico como aquel que abarca desde 75 minutos (la duración estimada de un partido) antes de que se quiera fijar el partido hasta 75 minutos después de la hora en la que se quiere fijar el partido. |
| **RF_Sprint_2.4** | Selección de canchas con pistas libres | El algoritmo debe devolver la selección de canchas, cuyo número de pistas sea **mayor exclusivamente** al número de partidos en dicha cancha dentro del horario crítico. |

Tabla de Requisitos 3.- Requisitos funcionales del *sprint* **Consultar Canchas Vacías**

### 4.3.2 Requisitos no funcionales

No se aporta ningún requisito extra a los ya mencionados anteriormente.

## 4.4 Fijar partido [SPRINT 3]

Se espera que la aplicación localice automáticamente a los rivales pertenecientes al grupo del equipo del usuario, y una vez seleccionado uno, permita fijar un partido, en una cancha determinada a una hora y una fecha concreta. Si el usuario que realiza la acción no tiene un rol de administrador, la acción no podrá ser llevada a cabo

### 4.4.1 Requisitos funcionales

| Nombre | Requerimiento | Descripción |
|---|---|---|
| **RF_Sprint_3.1** | Descartar usuarios por categoría | A esta funcionalidad solo deben poder acceder aquellos usuarios con una categoría del tipo "Administrador" |
| **RF_Sprint_3.2** | Obtener los equipos pertenecientes al grupo del equipo del usuario | Se deben obtener y listar los nombres e identificadores de los equipos que pertenecen al mismo grupo que el equipo del usuario que realiza la acción. |
| **RF_Sprint_3.3** | Descartar a los equipos con partidos ya fijados o jugados y al del usuario | Se descartarán de la lista anterior todos los equipos del grupo que ya hayan jugado o tengan un partido fijado contra el equipo del usuario. Asimismo se descartara de dicha lista al propio equipo del usuario. |
| **RF_Sprint_3.4** | Obtener datos completos de los equipos resultantes | Se deben obtener todos los datos, desde la base de datos de los equipos que aún continúan en la lista anterior (los que ni han jugado ni tienen partido fijado contra el equipo del usuario) y mostrarlos en la interfaz gráfica para que el usuario seleccione aquel contra el que quiere enfrentarse. |
| **RF_Sprint_3.5** | Seleccionar fecha y hora | Cuando el usuario seleccione a su rival, se le debe dar la opción de elegir la fecha y la hora en la que quiere jugar el partido. |
| **RF_Sprint_3.6** | Consultar canchas libres | Con la fecha y la hora obtenidas, se debe llamar al algoritmo del *sprint* 2, pasándole dichos datos, para que devuelva un listado con los datos de las canchas libres. Dicho listado tiene que mostrarse en la pantalla para que el usuario elija la cancha en la que quiere disputar el partido. |
| **RF_Sprint_3.7** | Actualizar base de datos | Una vez seleccionados todos los datos necesarios para fijar un partido (rival, fecha, hora y cancha), se procederá a guardar estos en base de datos para su posterior consulta. |
| **RF_Sprint_3.8** | Notificar a equipos implicados en el partido | Se debe notificar a los dos equipos implicados en el partido, mediante una notificación push (que será generada por la aplicación Android tras recibir un mensaje por GCM) de que el partido ha sido fijado indicando rival, fecha, hora y cancha. |

Tabla de Requisitos 4.- Requisitos funcionales del *sprint* **Fijar Partido**

### 4.4.2 Requisitos no funcionales

| Nombre | Requerimiento | Descripción |
|---|---|---|
| **RNF_Sprint_3.1** | Listados simples y aclarativos | Los listados que la aplicación debe mostrar, los de equipos y los de canchas, deben ser lo más simple posibles, informando al máximo al usuario. Por ejemplo, en el listado de rivales, con el nombre es suficiente, pero sin embargo, en el listado de canchas debe aparecer el nombre, la dirección, el transporte público sobre cómo llegar y los comentarios de otros usuarios al respecto de dicha cancha. |
| **RNF_Sprint_3.2** | Notificación informativa | La notificación debe ser suficientemente aclaratoria para el usuario, indicándole exactamente la fecha y hora, el rival y la cancha donde se disputara el partido. |

Tabla de Requisitos 5.- Requisitos no funcionales del *sprint* **Fijar Partido**

## 4.5 Actualizar resultado [SPRINT 4]

Se quiere que, para este *sprint*, la aplicación además de realizar el *login*, consultar las canchas vacías y fijar un partido, sea capaz de dado un partido y un resultado actualizarlo y realizar las diversas tareas que conllevan un cambio en el estado de un partido

### 4.5.1 Requisitos funcionales

| Nombre | Requerimiento | Descripción |
|---|---|---|
| **RF_Sprint_4.1** | Descartar usuarios por categoría | A esta funcionalidad solo deben poder acceder aquellos usuarios con una categoría del tipo "Administrador" |
| **RF_Sprint_4.2** | Obtener partido jugado | Cuando el usuario selecciona el partido que desea actualizar, se deben obtener todos los datos completos del partido |
| **RF_Sprint_4.3** | Introducir marcador | Se debe mostrar una nueva pantalla en la que el usuario pueda introducir el resultado del partido y este se debe guardar en memoria |
| **RF_Sprint_4.4** | Actualizar datos del partido | Se deben actualizar, en base de datos, la entrada relativa al partido que acaba de ser modificada por el jugador |
| **RF_Sprint_4.5** | Notificar a los equipos implicados | Se debe notificar a los dos equipos implicados en el partido, mediante una notificación push (que será generada por la aplicación Android tras recibir un mensaje por GCM) de que el partido ha sido actualizado, indicando el nuevo marcador, así como el rival de dicho partido. |

Tabla de Requisitos 6.- Requisitos funcionales del *sprint* **Actualizar Resultados**

### 4.5.2 Requisitos no funcionales

No se aporta ningún requisito extra a los ya mencionados anteriormente.

## 4.6 Consultar clasificación [SPRINT 5]

Al finalizar este *sprint* se espera que la aplicación sea capaz de realizar una consulta sobre la clasificación, y que sea capaz de mostrarla ordenadamente en una pantalla nueva, pudiendo seleccionar por grupo.

### 4.6.1 Requisitos funcionales

| Nombre | Requerimiento | Descripción |
|---|---|---|
| **RF_Sprint_2.1** | Modificar proceso que actualiza los partidos en base de datos | Se decide que el mejor punto para actualizar la clasificación, es tras actualizar los datos del partido en la base de datos, y a la vez actualizar la clasificación. Se actualizarán los datos según el resultado del partido. 2 puntos más a quien gane, 1 a quien pierda. Se actualizarán también las canastas a favor y en contra y los partidos ganados o perdidos según el resultado. |
| **RF_Sprint_2.2** | Obtener clasificación dado un grupo | Dado un grupo, se tiene que obtener un listado de los equipos de ese grupo con los datos de la clasificación (puntos, victorias, derrotas, canastas a favor, canastas en contra) y mostrarlos en la interfaz gráfica. |

Tabla de Requisitos 7.- Requisitos funcionales del *sprint* **Consultar Clasificación**

### 4.6.2 Requisitos no funcionales

No se aporta ningún requisito extra a los ya mencionados anteriormente.

## 4.7 Consultar partidos [SPRINT 6]

Con esta tarea, se pretende que el usuario final, a través de la aplicación pueda consultar sus partidos. Se pretende que se puedan consultar tantos los partidos jugados como los que han sido fijados, pero aún no se han jugado. Debe facilitarse la información en una vista de tal manera, que, si se desea, en el futuro, al pulsar sobre cada partido, se pueda realizar la acción requerida.

### 4.7.1 Requisitos funcionales

| Nombre | Requerimiento | Descripción |
|---|---|---|
| **RF_Sprint_6.1** | Obtener listado de partidos fijados y jugados del usuario | Se debe obtener el listado de partidos tanto jugados como fijados y separarlos en dos listas para su posterior manejo por parte de la aplicación Android. |
| **RF_Sprint_6.2** | Obtener los datos de los partidos | Se debe obtener y actualizar ambos listados con los datos completos de cada partido. De los partidos fijados se obtendrá la cancha, la fecha y la hora. De los partidos jugados se obtiene el resultado del partido. Además, debe crearse una nueva vista para mostrar ambos listados para que el usuario pueda interactuar con ellos. |

Tabla de Requisitos 8.- Requisitos funcionales del *sprint* **Consultar Partidos**

### 4.7.2 Requisitos no funcionales

No se aporta ningún requisito extra a los ya mencionados anteriormente.

## 4.8 Consultar calendario [SPRINT 7]

En este momento, la aplicación debe ser capaz de mostrar el calendario de partidos fijados, desde el momento actual en adelante. Esta operación podrá ser consultada por todos los usuarios de la aplicación, indistintamente del rol que posean dichos usuarios.

### 4.8.1 Requisitos funcionales

| Nombre | Requerimiento | Descripción |
|---|---|---|
| **RF_Sprint_7.1** | Obtención partidos fijados | Se debe obtener un listado completo con todos los partidos fijados de la base de datos. |
| **RF_Sprint_7.2** | Descartar partidos anteriores a la fecha actual | Se debe descartar del listado de partidos fijados aquellos cuya fecha de partido sea anterior a la fecha actual. No debería darse este caso, pero podría haber partidos que se hayan jugado ya, pero no se hayan actualizado, y por tanto siguen marcados como partido fijado. |

Tabla de Requisitos 9.- Requisitos funcionales del *sprint* **Consultar Calendario**

### 4.8.2 Requisitos no funcionales

No se aporta ningún requisito extra a los ya mencionados anteriormente.

## 4.9 Obtener información sobre las canchas [SPRINT 8]

Debido al formato peculiar de la liga, se desea mantener un control sobre las canchas en las que se disputan los partidos. Actualmente existe un dossier con todas las canchas, consultable por cualquier persona que quiera. Al finalizar esta tarea, dicho dossier, debe ser también electrónico y disponible desde la aplicación. Se debe diseñar de tal manera que, en un futuro, si se desea llevar a cabo, se pueda realizar una búsqueda y un filtrado por ciertos datos de las canchas, como el barrio, la dirección, el nombre o la línea de metro.

### 4.9.1 Requisitos funcionales

| Nombre | Requerimiento | Descripción |
|---|---|---|
| **RF_Sprint_8.1** | Obtener listado de canchas | Se debe obtener un listado de todas las canchas existentes en la base de datos. |
| **RF_Sprint_8.2** | Obtener información completa de canchas | Se debe obtener los datos completos de todas las canchas existentes en la lista anterior. Los datos que deben descargarse son el nombre, la dirección, la parada de metro, la parada de autobús más cercana, si dispone de iluminación, el horario de la iluminación, la URL de las imágenes y la información adicional de las canchas. |
| **RF_Sprint_8.3** | Descargar imágenes de las canchas | Se deben descargar las imágenes correspondientes a las URL de todas las canchas |
| **RF_Sprint_8.4** | Actualizar vista de las canchas | Con el listado de canchas y sus datos, y las imágenes descargadas, se debe montar una nueva vista en la que aparezcan las canchas y sus fotos. |

Tabla de Requisitos 10.- Requisitos funcionales del *sprint* **Obtener Información Canchas**

### 4.9.2 Requisitos no funcionales

| Nombre | Requerimiento | Descripción |
|---|---|---|
| **RNF_Sprint_8.1** | Uso de Cards de Material Design | El listado de canchas, con su información y su respectiva fotografía debe aparecer sobre un listado de cards, una de las nuevas funcionalidades de Material Design. |

Tabla de Requisitos 11.- Requisitos no funcionales del *sprint* **Obtener Información Canchas**

## 4.10 Obtener carteles de la liga [SPRINT 9]

Los carteles, para esta liga, son el medio de expansión más importante, pues permiten dar a conocer la liga y sus eventos a una gente que no le llegaría de no ser por ellos. Por eso mismo se solicita crear una tarea que permita mostrar al usuario todos los carteles de la liga ofreciendo una información más detallada del mismo al pulsar sobre él. Se trata de una tarea muy similar a la anterior, que comparte el número de requisitos y el comportamiento básico de cada uno de ellos, centrándose en este caso, en los carteles de la liga.

### 4.10.1 Requisitos funcionales

| Nombre | Requerimiento | Descripción |
|---|---|---|
| **RF_Sprint_9.1** | Obtener listado de carteles | Se debe obtener un listado de todos los carteles almacenados en la base de datos. |
| **RF_Sprint_9.2** | Obtener información completa de carteles | Se debe obtener los datos de todos los carteles de la lista anterior. Los datos que se deben obtener son el título del cartel, la URL de la imagen y los datos informativos del cartel. |
| **RF_Sprint_9.3** | Descargar imágenes de los carteles | Se deben descargar las imágenes correspondientes a las URL de todos los carteles |
| **RF_Sprint_9.4** | Actualizar galería de los carteles | Se crea una galería dinámica con todas las imágenes de los carteles obtenidos de la base de datos. Al pulsar sobre cada cartel, se debe abrir una nueva ventana con el título, la información y la imagen del cartel pulsado. |

Tabla de Requisitos 12.- Requisitos funcionales del *sprint* **Obtener Carteles Liga**

### 4.10.2 Requisitos no funcionales

No se aporta ningún requisito extra a los ya mencionados anteriormente.

# 5. Diseño e implementación

En este quinto apartado de la memoria, se tratará la parte del diseño de la aplicación. Aquella en la que se idea como será finalmente la aplicación, diseñando tanto la interfaz gráfica o *GUI* como las entidades y las relaciones que formaran la base de datos, las clases de la aplicación y la distribución de las funciones de *PHP*.

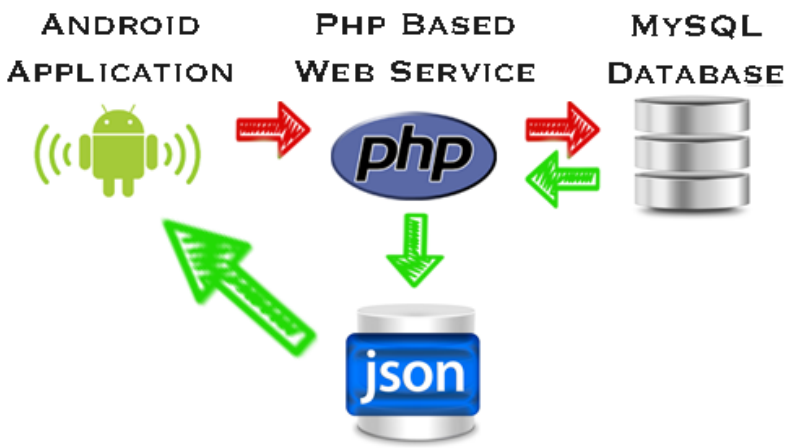



Imagen 12.- Diagrama representativo de la aplicación desarrollada. Fuente: https://www.fiverr.com/anilsapovadiya/do-json-mysql-api-for-android-and-iphone

Como se observa en **Imagen 12**, y a pesar de que en Android es complicado separar claramente las tres capas que propone el patrón *MVC* que son el modelo (datos) la vista (la interfaz para el usuario) y el controlador (encargado de suministrar los datos a la vista y que el origen de estos resulte invisible para la vista) ya que la vista y el controlador están acoplados mutuamente e imposibilita la abstracción de las capas, el resultado obtenido, es claramente un modelo vista controlador, con algunas peculiaridades que se comentaran más adelante.

Este apartado se dividirá en 4 apartados. Tres de ellos dedicados a cada una de las capas que forman parte del patrón *MVC*, y el cuarto dedicado a los diagramas que se han desarrollado para facilitar el entendimiento del diseño de la aplicación.

## 5.1 Modelo

En este epígrafe se desarrollará todo aquello que tiene que ver con el modelo, es decir, los datos. Por tanto, este apartado estará dedicado exclusivamente a la base de datos *MySQL* [3].

Se utiliza una única base de datos llamada **ligacooperativabasket** cuyo esquema relacional es el que aparece en la Imagen 13.

Cada una de estas tablas ha ido creándose y relacionándose con el resto según avanzaban los distintos *sprints* de la aplicación. El momento de creación ha sido durante el Login [SPRINT 1] las tablas **usuarios**, **roles**, **equipo** y **sessions**. Durante el Consultar canchas vacías [SPRINT 2] las tablas **canchas** y **partidos**. Durante el Consultar clasificación

[SPRINT 5] se crea la tabla **clasificación**. En el Obtener información sobre las canchas [SPRINT 8] se crea la tabla **images**.

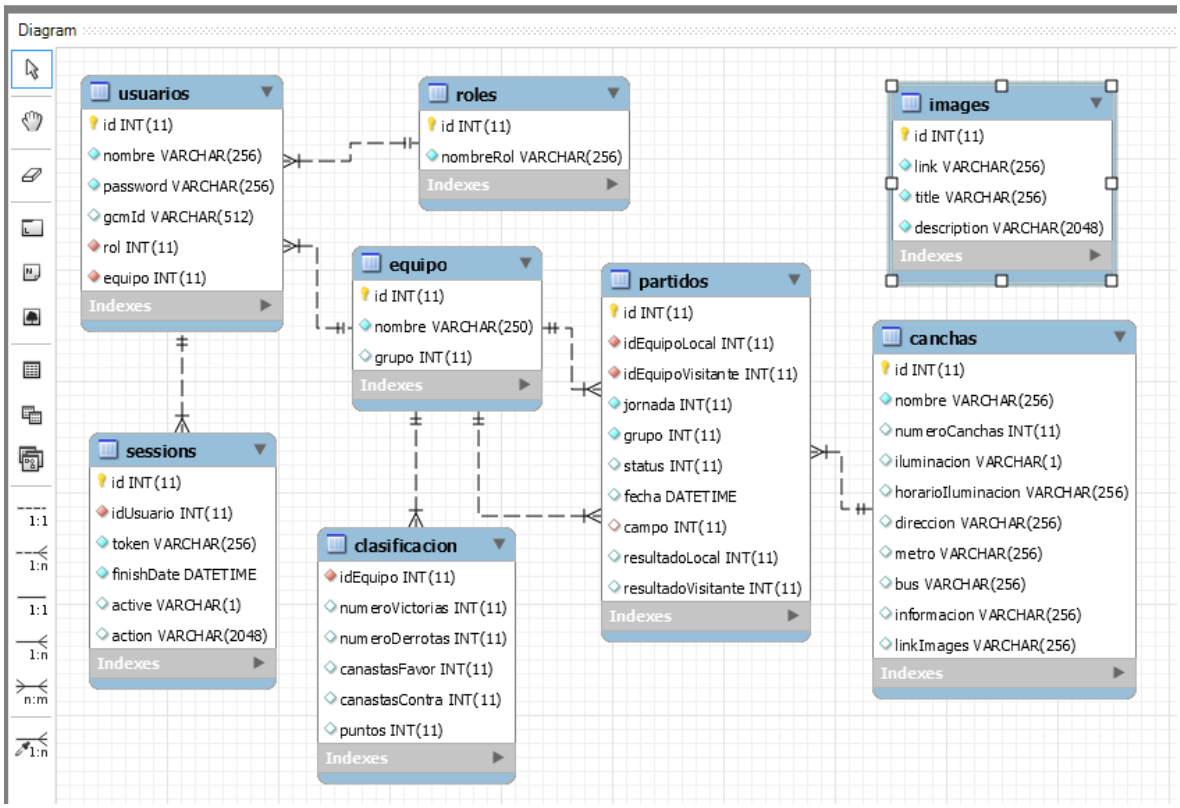



Imagen 13.- Esquema relacional de la base de datos ligacooperativabasket. Fuente: Propia

En la tabla **usuarios** se almacenan los datos de cada una de las personas que tienen acceso a la aplicación, ya sea con rol de administrador o con rol de usuario. Esta tabla tiene referenciado el rol del usuario y el equipo al que pertenece. El esquema de la tabla es el siguiente.

| *Login* | | |
|---|---|---|
| Campo | Tipo | Descripción |
| id | int | Identificador del usuario. |
| nombre | varchar(256) | Nombre del usuario dentro de la aplicación |
| password | varchar(256) | Contraseña de acceso a la aplicación de este usuario |
| gcmId | varchar(512) | Identificador del dispositivo móvil valido para (GCM) |
| rol | int | Identificador del rol del usuario |
| equipo | int | Identificador del equipo al que pertenece el usuario |

**Tabla 1.- *Login***

En la tabla de **roles** se almacenan los tres tipos de roles que puede adquirir un usuario. Estos son **Administrador**, **Usuario** o **Visitante**. Cada uno de los roles da acceso a una determinada funcionalidad de la aplicación. El Administrador tiene acceso a toda la aplicación, podría decirse que tiene accesos de lectura y escritura (puede realizar acciones de consulta sobre las tablas, y acciones de modificación e inserción de los datos de las tablas). El Usuario tiene accesos únicamente de lectura, sobre todas y cada una de las

tablas, y el Visitante que tiene accesos únicamente de lectura sobre determinadas tablas que no exponen nada a gente que no pertenezca a la liga. Su esquema es el siguiente.

| Roles | | |
|---|---|---|
| Campo | Tipo | Descripción |
| id | int | Identificador del rol de los usuarios |
| nombreRol | varchar(256) | Nombre del rol de los usuarios |

**Tabla 2.- Roles**

En la siguiente tabla, la tabla de **equipo**, se almacenan los datos de todos los equipos que participan en el desarrollo de la liga. En esta edición se cuenta con 48 participantes, los cuales tienen un nombre, se les asigna un grupo y al inscribirse en la base de datos, esta, automáticamente les asigna un valor de id. Su esquema es el siguiente

| Equipo | | |
|---|---|---|
| Campo | Tipo | Descripción |
| id | int | Identificador del equipo |
| nombre | varchar(256) | Nombre del equipo |
| grupo | int | Grupo del equipo dentro de la liga |

**Tabla 3.- Equipo**

En la tabla **sessions** se almacenan los datos de las sesiones de los usuarios activos en la aplicación, es decir, los que están en dicho momento conectados. Esta tabla sirve a su vez de log para los desarrolladores pues va almacenando las trazas del flujo de los usuarios dentro de la aplicación. En ella se almacena el identificador de usuario, el *token* de *session* que corresponde a una cadena alfanumérica que sirve para identificar el usuario que realiza las acciones sobre la base de datos después del *login*. Se almacena un valor de una fecha futura que corresponde al momento en el que dicho *token* caduca. Está predefinido que el *token* caduque a los diez minutos, pero con cada nueva acción de consulta/modificación de la base de datos, el *token* actual es autocaducado y se asigna un nuevo *token* con un nuevo valor de caducidad. A su vez, para un mayor control de las sesiones se almacena un valor de *active* que corresponde a si ese *token* es el actual o no. Cuando el *token* es el actual, dicho valor es un **1**, sin embargo, si se realiza una nueva acción, y este *token* se autocaduca, a su vez a active se le otorga el valor de **0**. En *action* almacena un texto referente a la acción realizada por el usuario cuando se le asigna el *token* correspondiente

| Sessions | | |
|---|---|---|
| Campo | Tipo | Descripción |
| id | int | Identificación del token al que pertenece dicha sesión. |
| idUsuario | int | Identificador del usuario de la sesión. |
| token | varchar(256) | Token de la sesión. |
| finishDate | DateTime | Fecha de caducidad del token y por tanto de la sesión |
| active | varchar(1) | Identifica si el token al que pertenece está activo o no |
| action | varchar(2048) | Acción realizada por el usuario al iniciar la sesión. |

**Tabla 4.- Sessions**

Hasta aquí lo que serían las tablas referentes al primero de los *sprints* realizados. Las dos siguientes tablas en ser analizadas corresponderán al segundo *sprint*.

En la tabla **partidos** se almacena la información relativa a todos los partidos que tienen que disputarse en la fase actual. Al inicio de cada fase, se introducen en la tabla, todos los partidos de dicha fase, sin resultado ni fecha ni hora ni lugar, simplemente los equipos que participan, la jornada teórica a la que pertenece el partido y el grupo de los equipos que juegan. Más adelante, cuando un partido se fija o se juega se modifica la entrada correspondiente añadiendo los datos correspondientes. Los datos que se modifican son el *status* que pasa de tener un valor 0 (estado inicial) a un valor 1 o 2 ya sea un partido fijado o jugado respectivamente. Cambia la fecha, el campo y el resultadoLocal y resultadoVisitante.

| Partidos | | |
|---|---|---|
| Campo | Tipo | Descripción |
| id | int | Identificador del partido |
| idEquipoLocal | int | Identificador del equipo local |
| idEquipoVisitante | int | Identificador del equipo visitante |
| jornada | int | Número de la jornada |
| grupo | int | Numero de grupo de los equipos participantes |
| status | int | Indica el estado del partido. |
| fecha | DateTime | Fecha del partido con *status* igual a 1 o 2. Null en caso contrario. |
| campo | int | Identificador del campo con *status* igual a 1 o 2. Null en caso contrario. |
| resultadoLocal | int | Canastas del local con *status* 2. Null en caso contrario. |
| resultadoVisitante | int | Canastas del visitante con *status* 2. Null en caso contrario. |

**Tabla 5.- Partidos**

En la tabla **canchas** se almacenan todos los datos de las canchas, desde su dirección, el nombre, el transporte público para llegar, así como una URL donde se almacena una foto de cada cancha. A la hora de descartar canchas por problemas lumínicos, el campo utilizado es **iluminación**

| Canchas | | |
|---|---|---|
| Campo | Tipo | Descripción |
| id | int | Identificador de la cancha |
| nombre | varchar(256) | Nombre de la canchas |
| numeroCanchas | int | Numero de pistas de baloncesto en la cancha |
| iluminación | varchar(1) | Indica si existe iluminación adicional o no. |
| horarioIluminacion | varchar(256) | Horario de iluminación de la cancha, si procede |
| dirección | varchar(256) | Calle y número de la cancha |
| metro | varchar(256) | Parada de metro más cercana |
| bus | varchar(256) | Línea de autobús más cercana |
| información | varchar(256) | Informaciones extra sobre la cancha. |
| linkImages | varchar(256) | URL de la imagen de la cancha |

**Tabla 6.- Canchas**

La tabla de **clasificación** del *sprint* 5 contiene un listado de todos los equipos en los que tras cada partido jugado se va actualizando las entradas correspondientes añadiéndole los puntos ganados por cada equipo, así como las canastas a favor y en contra y se actualizarán por último los partidos ganados y perdidos y los puntos de los equipos implicados en el partido

| Clasificación | | |
|---|---|---|
| Campo | Tipo | Descripción |
| idEquipo | int | Identificador del equipo. |
| numeroVictorias | int | Número de victorias del equipo |
| numeroDerrotas | int | Numero de derrotas |
| canastasFavor | int | Canastas metidas por el equipo |
| canastasContra | int | Canastas recibidas por el equipo |
| puntos | int | Puntos del equipo |

**Tabla 7.- Clasificación**

La ultima tabla en ser creada, durante el *sprint* 8, es la tabla de **images**, en la que se almacenan los datos relativos a los carteles informativos de la liga y otros colectivos relacionados con la liga de una manera u otra

| Images | | |
|---|---|---|
| Campo | Tipo | Descripción |
| id | int | Identificador del cartel. |
| link | varchar(256) | Link de la imagen del cartel |
| title | varchar(256) | Título del cartel |
| description | varchar(2048) | Información general del cartel. |

**Tabla 8.- Images**

## 5.2 Controlador

Este apartado se centra en la capa del controlador, que es aquella encargada de solicitar los datos al modelo y maquetarlos para entregárselos a la vista de tal manera que esta pueda exponerlos, resultándole invisible la manera de conseguirlos o el origen de dichos datos. Con esta separación se pretende lograr una abstracción total, para que, si en el día de mañana, el modelo cambia, o el formato en el que el controlador tiene que obtener los datos, no haya que modificar también la vista. Por tanto, el controlador es la capa con más importancia y más trabajo dentro del patrón *MVC*.

Como se comentó anteriormente, se considera que existen dos controladores dentro de la aplicación, debido a la dificultad de *Android* de separar vista y controlador. Como se puede observar en Imagen 12, la única capa que consulta sobre base de datos es la capa web de PHP, por tanto se pude considerar que es el controlador más parecido a los denominados controladores en los proyectos MVC puro. Sin embargo, Android también posee su capa de controlador, pues, a su vez, se encarga de traducir los datos devueltos por la capa web a un formato que la vista Android sea capaz de dibujar.

Por tanto, se observa que existen dos patrones *MVC* dentro del mismo proyecto. El primero formado por la aplicación Android que sería la vista, la capa web (*PHP*) el controlador y la base de datos *MySQL* el modelo, este se sustenta en que la capa web solicita datos al modelo, y se los devuelve a la vista, y para el controlador son invisibles las acciones que debe realizar la vista para mostrar los datos. Y otro *MVC*, este, desde el punto de vista de la aplicación *Android* únicamente, según el cual, dentro de la aplicación estarían la vista (*XML*) y el controlador (*Java*) y la capa web será el modelo, puesto que la capa de controlador (*Java*) solicita datos a la capa web y este le devuelve unos datos, que él tiene que transformar para la vista.

### 5.2.1 Controlador Web (PHP)

El controlador formado por la capa web, que será desarrollada en el lenguaje PHP [5] estará encargado de recibir datos de la aplicación Android en formato *JSON,* realizar consultas sobre la base de datos, y devolver los datos obtenidos en formato *JSON* de la misma manera que le llegaron.

Para fortalecer la seguridad de la aplicación, como se incluye en algunos de los requisitos expresados en el apartado anterior, en vez de realizar consultas directas contra la base de datos, se hace uso de las *prepared statements* que aparte de favorecer la seguridad incrementan la eficiencia de las consultas en base de datos.

El funcionamiento de las *prepared statements* [4] pasa por tres fases distintas que son la **preparación** (en la que se crea la *prepared statement* y se envía a la base de datos, teniendo en cuenta que los parámetros de la consulta deben ir vacíos, del estilo a INSERT INTO table VALUES (?, ?, ?)), la **compilación** (en la que la base de datos parsea, compila y realiza optimizaciones de la quiere y guarda los resultados sin ejecutar la *query*) y la **ejecución** (en la que la aplicación, haciendo uso de la función ***bindParams()*** añade los valores de los parámetros a la *query* y la base de datos ejecuta el *prepared statement*).

Los beneficios de estas son la reducción de tiempo de parseo, y la reducción del número de veces que se prepara la *query*, pues esta solo se prepara una vez a pesar de ejecutarse múltiples veces. La vinculación (*bind*) de parámetros reduce el ancho de banda usado entre el servidor y la aplicación, pues solamente se necesita enviar los parámetros y no toda la *query*. Y el último beneficio, como se comentó anteriormente, es que aumenta la seguridad, pues son muy efectivos frente a los ataques mediante *sql injection* debido a que usan otro protocolo y no necesitan introducir caracteres de escape frente a caracteres especiales de *SQL* [2].

Por si acaso, de todas maneras, se ha tenido especial cuidado en la entrada de datos por parte del usuario, comprobando todas y cada una de ellas y realizando comprobaciones para evitar la entrada de caracteres no deseados, como podrían ser comillas (‘), caracteres de comentario (--) o puntos y coma (;)

### 5.2.2 Controlador app (Android)

Como se ha comentado anteriormente, existen dos capas de controlador, la web, y la de la aplicación *Android*. En este caso nos centramos en la capa de la aplicación *Android*.

Para la aplicación *Android*, la capa web actúa como modelo del patrón *MVC*, con lo que eso conlleva, es decir, la aplicación realiza una solicitud de datos a su “modelo” en formato *JSON*, y sin saber cómo ocurre, los datos solicitados le llegan en el mismo formato que los solicitó, en formato *JSON*.

Tras recibir los datos, esta capa del controlador, se encarga de mapearlos de tal manera que la vista (dentro de la propia aplicación) sea capaz de poder dibujarlos o mostrarlos en la *GUI* del usuario.

Debido a que se trata de un proyecto bajo la metodología *agile*, el diagrama de clases realizado durante el diseño fue creciendo según se realizaban nuevos *sprints* en el proyecto. Esto es debido, a como se explicó anteriormente, la integración continua de

*Scrum*. Para no añadir los 9 diagramas a la memoria se ha decidido mostrar únicamente el diagrama de clases de último *sprint* que a su vez corresponde a la aplicación completa.

El diagrama de clases, que se muestra en el anexo **D**, ha sido realizado con la herramienta *IntelliJ IDEA* con el código ya desarrollado. Se corresponde casi a la perfección con el creado durante las fases de diseño, y las diferencias serán explicadas en los apartados correspondientes

En Imagen 14 se puede observar la distribución, en un listado, de las clases Java en sus paquetes correspondientes, y en la Imagen 20. Se puede observar el diagrama de clases de la aplicación *Android* y como estas se relacionan entre sí, en un diagrama similar a los diagramas entidad-relación de las bases de datos.

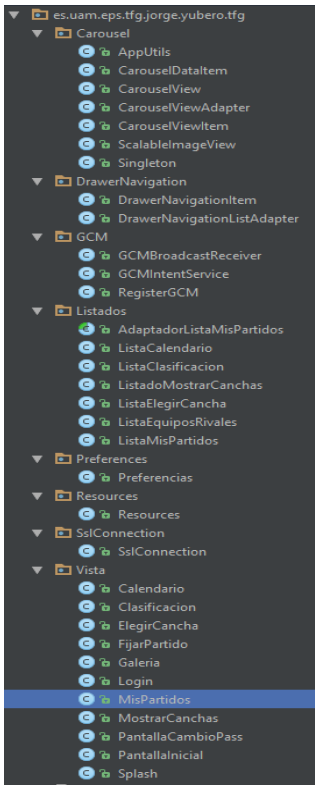



Imagen 14.- Listado de clases y paquetes. Fuente: Propia

Como se puede observar, está el paquete ***Vista*** en el que se alojan todas las clases que corresponden a alguna de las vistas de la aplicación. Todas ellas están relacionadas con alguno de los *sprints* de los que se compone este proyecto.

Al primer *sprint* le corresponden la clase ***Login***, la clase ***Splash*** y la clase ***PantallaCambioPass***. Al segundo *sprint* no le corresponde ninguna vista, pues se trata de un algoritmo únicamente funcional, el cual no muestra nada al usuario. Al tercer *sprint* le corresponden las clases ***FijarPartido*** y la clase ***ElegirCancha*** (la cual hace uso del algoritmo del segundo *sprint* para obtener sus datos)

Al cuarto *sprint* no le corresponde ninguna de las vistas actuales, pues a lo largo del sexto *sprint* y como consecuencia de la integración de las tareas, la vista de este *sprint* se sustituyó por la vista del sexto *sprint* para englobarlo de manera que resulte más intuitivo para el usuario.

Al quinto *sprint* le corresponde la clase ***MostrarClasificacion***. A su vez, al sexto, le corresponde la clase ***MisPartidos*** (la remarcada en azul en la imagen), la cual sustituye a la vista del cuarto *sprint*, puesto que anteriormente existía una vista en la que, dados los partidos, se seleccionaban, y se actualizaba el resultado, se decidió integrarlo dentro de la vista de MisPartidos, en la que aparecen tanto los partidos fijados como los jugados.

Al séptimo, octavo y noveno *sprints* le corresponden las clases **Calendario**, **MostrarCanchas** y **Galería** respectivamente.

Para el ultimo *sprint*, el noveno, como se puede observar en el diagrama de clases completo del anexo **D**, no solo se crea la clase Galería, sino que se hace uso de un paquete entero para lograr la galería en la que se muestran los carteles de la liga (galería que será mostrada en el apartado relacionado con las maquetas de la aplicación). Dicho paquete es el paquete ***Carousel*** con sus clases ***AppUtils***, ***CarouselDataItem***, ***CarouselView***, ***CarouselViewAdapter***, ***CarouselViewItem***, ***ScalableImageView*** y ***Singleton***. Este paquete es una de las diferencias con el diagrama de clases original, pues en el original, se había

ideado que este conjunto de clases estuviese formado por una sola clase con una galería más sencilla que la finalmente elegida. Tras indagar por internet, se encontró esta modelo de galería, que finalmente ha sido la elegida, y se modificaron los planes iniciales respecto al diagrama de clases inicial.

El paquete ***DrawerNavigation*** contiene las clases ***DrawerNavigationItem*** y ***DraweNavigationListAdapt***, que como se puede ver en el diagrama de clases todas las clases del paquete ***Vista*** hacen uso de ellas, esto es debido a que se corresponden a un menú lateral que aparece en todas las vistas de la aplicación. Este menú, actúa a modo de índice y redirige al usuario a la pantalla que seleccione.

En el paquete ***GCM*** se alojan las clases ***GCMBroadcastReceiver***, ***GCMIntentService*** y ***RegisterGCM***, las cuales son las encargadas del envío y la recepción de los mensajes con el servidor de Google para la mensajería (*Google Cloud Messaging* de sus siglas ***GCM***).

En el paquete ***Listados*** se almacenan todas aquellas clases de las que hacen uso las vistas que necesitan mostrar una lista en ellas, como puede ser la vista de la clasificación. Estas clases son ***AdaptadorListaMisPartidos***, ***ListaCalendario, ListaClasificacion, ListadoMostrarCanchas, ListaElegirCancha, ListaEquiposRivales*** y ***ListaMisPartidos***.

Las clases ***Preferencias, Resources*** y ***SslConnection*** son las otras diferencias existentes entre el diagrama de clases original y el actual, pues se ideo que dichas clases no existiesen y el código en ellas estuviese repartido entre las diversas clases, pero a mitad del desarrollo se decidió independizarlo en clases diferentes para favorecer la reutilización de código.

## 5.3 Vista

En este apartado, se hablará sobre la última capa que queda por definir del patrón *MVC*, la vista. Esta capa corresponde a la *GUI* de las siglas del inglés *Graphical User Interface*, es decir, a la interfaz gráfica del usuario, la encargada de recoger todas las acciones que realiza el usuario, enviárselas a su controlador para que este actúe en consecuencia a las acciones solicitadas

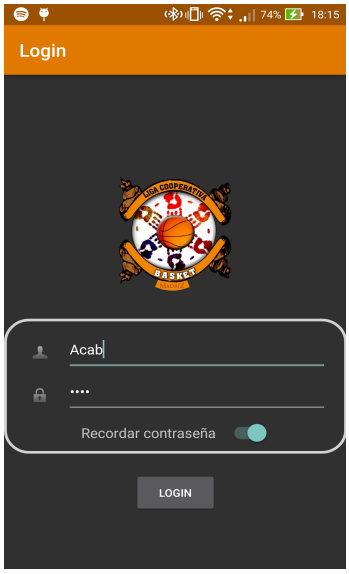
Se implementa en la aplicación *Android*, y se codifica bajo el lenguaje de etiquetas ***XML***. Se trata de definir una vista sencilla y agradable al uso, según los requerimientos de los usuarios finales.

Se trata de usar la última tecnología de *Android* disponible al inicio del desarrollo, que es su versión 5.1.1 del 19 de abril de 2015, aunque pocos meses después llegaría la versión 6.0 el 5 de octubre de 2015. Esto conlleva el uso de *Material Design* para la vista. *Material Design* basa su estilo en la máxima expresión de la sencillez posible. Se busca acabar con todas las decoraciones extras y se apuesta por el uso de figuras geométricas tales como el circulo o los rectángulos de todo tipo.
Siguiendo la idea de *Google* del uso de *Material Design*, se ha desarrollado toda la aplicación, como se mostrará en las maquetas del siguiente apartado.

Debido a la metodología utilizada a lo largo del desarrollo del proyecto y como viene haciéndose a lo largo de esta memoria, la capa perteneciente a la vista también se ha ido desarrollando según los distintos *sprints* por los que ha pasado el proyecto.

### 5.3.1 Maquetas

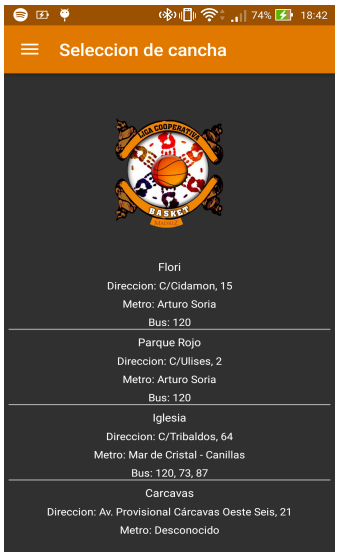
Aquí se mostrarán las maquetas usadas para orientar a los desarrolladores en el momento de codificar el proyecto, concretamente a la hora de codificar la interfaz gráfica. Como ocurriese con el diagrama de clases de la capa del controlador en la aplicación Android, para mejorar la visualización y la realización de estas maquetas, las mostradas a continuación corresponden a las pantallas finales de la propia aplicación, obtenidas a través de capturas de pantalla del propio dispositivo.



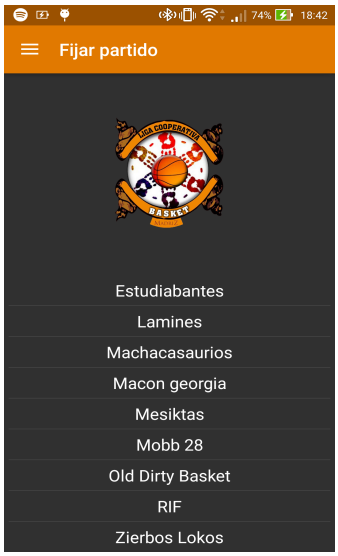

Maqueta 1.- Fuente: Propia


Maqueta 2.- Fuente: Propia


Maqueta 3.- Fuente: Propia




Maqueta 4.- Fuente: Propia




Maqueta 5.- Fuente: Propia


Maqueta 6.- Fuente: Propia

## 5.4 Diagramas

En este último epígrafe del apartado referente al diseño de la aplicación, se expondrán algunos diagramas realizados durante esta fase del proyecto. Como ocurría con casos anteriores, solo se expondrán diagramas del último *sprint* pues este último, debido a la integración continua de la metodología *Scrum* contiene a todos los anteriores.

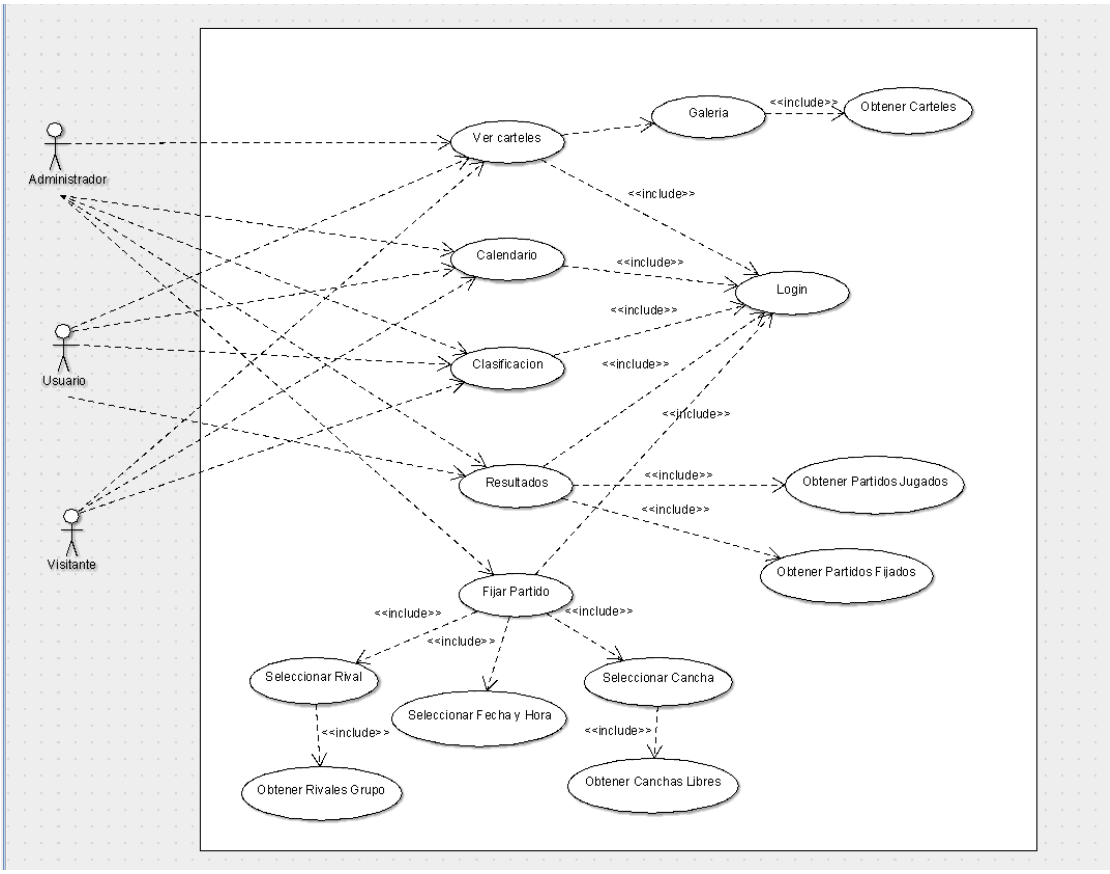



Imagen 15.- Diagrama de casos de uso de la aplicación. Fuente: Propia

En la Imagen 15 se muestra el diagrama de casos de uso de algunas de las funciones más utilizadas de la aplicación. Como se puede observar, son cinco las funciones principales que son **Ver carteles**, **Calendario**, **Clasificación**, **Resultados** y **Fijar Partido**. Todas ellas, hacen uso de la función *Login*, y eso se observa en el diagrama, con las flechas con la leyenda <<include>> que salen de cada una de las cinco funciones hacia el globo que pone ***Login***.

Se puede observar a su vez que el usuario con rol **Administrador** puede hacer uso de las cinco funciones de la aplicación, mientras que el usuario con el rol **Usuario**, únicamente puede hacer uso de aquellas que no conllevan una modificación en la base de datos, es decir, **Ver Carteles, Calendario, Clasificación** y **Resultados**.

A su vez, el usuario con el rol **Visitante**, es decir, aquellas personas ajenas a la liga que se interesan de algún modo de ella, únicamente podrán acceder a tres funciones claves que son **Ver Carteles**, puesto que esta es la mayor referencia de publicidad sobre la liga, la función **Clasificación y** la función **Calendario**.


Imagen 16.- Diagrama de flujo de la aplicación para fijar un partido. Fuente: Propia

Como se puede observar en Imagen 16 el proceso a seguir es primeramente realizar el ***Login***, a continuación, desde la pantalla inicial se pulsa el botón **Fijar Partido** que redirige a la pantalla de **Selección de Rival**. Una vez seleccionado, se muestra un *pop-up* para seleccionar la fecha del partido y otro después para la hora. Cuando rival, fecha y hora están elegidos, el algoritmo de buscar cancha libre entra en acción y muestra al usuario las canchas disponibles para esa fecha y esa hora en la pantalla **Seleccionar Canchas**. Una vez seleccionada una de las canchas disponibles, el usuario es redirigido a la pantalla inicial.

# 6. Desarrollo

Tras la fase de análisis y el posterior diseño, llega el momento de comenzar la codificación de la aplicación, ya sea la capa web, la capa de aplicación *Android*, o la capa de base de datos.

Tanto en la capa web, como en la capa del cliente *Android* se ha escrito un código legible y autoexplicativo, tratando de prescindir de los comentarios lo máximo posible, pero no olvidándose de ellos en zonas críticas del código. Se han utilizado nombres de variables siguiendo la nomenclatura húngara, según la cual, el nombre de una variable debe estar compuesto por un prefijo que indique su tipo de datos, en minúscula, y el nombre de la funcionalidad que dicha variable debe cumplir.

Como se explicó en apartados anteriores, el código del cliente Android se ha dividido en paquetes en los que se engloban las clases que comparten una funcionalidad similar.

Uno de los paquetes utilizados, el paquete denominado **vistas** contiene las clases correspondientes a todas las vistas de la aplicación, desde el *Login* hasta la pantalla de consultar las canchas de la liga. Cada una de estas clases está directamente relacionada con cada uno de los *sprints* que se iban realizando.

En la capa web, las distintas funcionalidades que se iban creando con el avance del proyecto y los *sprints* se iban organizando en diferentes ficheros PHP que se iban alojando en el servidor web.

Por su parte, lo más destacable del código PHP, es el algoritmo que se utiliza para devolver el listado de canchas libres dada una hora y una fecha. En un principio, este algoritmo se diseñó para que solicitase a la base de datos la lista de todos los partidos previstos, así como las canchas de las que se tiene constancia en la base de datos.

En la Imagen 17 se describe el flujo de ejecución del algoritmo utilizado.

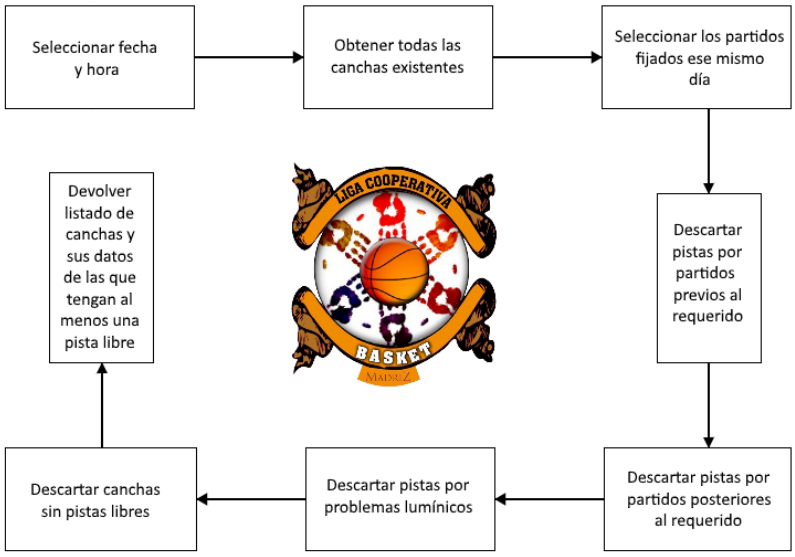



Imagen 17.- Diagrama de flujo seguido por el algoritmo en su ejecución.
Fuente: Propia

# 7. Integración, pruebas y resultados

Como se ha comentado con anterioridad, a lo largo de todo el documento, el uso de la metodología *agile*, en concreto la metodología *Scrum*, la integración, durante el desarrollo de la aplicación, la capa web, y la base de datos es continua, provocando que al final de cada uno de los *sprints* se realice una entrega al cliente (en este caso a un grupo de jugadores de la liga encargados de realizar el testing, en adelante, **el cliente**) de una aplicación completamente funcional que cumpla los requisitos establecidos para ese *sprint*, pero también, los establecidos para todos los *sprints* realizados con anterioridad al actual.

Este tipo de metodología implica que la integración no sea total, una vez acabado cada uno de los módulos de la aplicación, sino que se lleva a cabo parcialmente, según se van acabando los módulos o *sprints* en este caso.

A su vez, el banco de pruebas, se realiza antes, durante y después de la entrega tras cada uno de los *sprints*. Antes de la entrega se realizan las pruebas unitarias a la funcionalidad nueva del *sprint* actual. Durante la entrega, se realizan las pruebas de integración, que corren a cargo tanto del cliente como de los desarrolladores. Y tras la entrega, se realizan las pruebas de funcionalidad complementarias, que son aquellas encargadas de ver, que a pesar de que la aplicación funciona tras instalarle el nuevo módulo, no ha perdido ninguna de las funcionalidades que poseía con anterioridad a la entrega de este *sprint*.

Cualquier *bug* o error detectado en cada una de las tres pruebas, conlleva una nueva iteración del *sprint* actual, al que se le añade como tareas de *backlog* el arreglar dichos bugs o errores. Encontrar errores durante estas pruebas tiene como inconveniente que retrasa el inicio, y por tanto la entrega del siguiente *sprint*, pero tiene como beneficio, que los fallos se tiene más o menos localizados y se evita su expansión en futuras entregas, además, de que se cumple una de las normas básicas de la metodología *agile*, que es, que un *sprint* se da por terminado cuando este es completamente funcional y sin error alguno detectado.

Las pruebas unitarias, llevadas a cabo por el desarrollador, han consistido en tres tipos de pruebas: las pruebas de la capa web, las pruebas de la capa de base de datos y las pruebas de la aplicación Android.

Las pruebas sobre la capa **web** han consistido en la creación de páginas webs muy simples formadas por formularios en los que se solicitaban los datos necesarios para cada una de las nuevas funcionalidades, y un cuadro de texto en el cual aparecía el resultado. El banco de pruebas realizado sobre esta capa ha sido siempre el mismo.

- La primera prueba consistía en introducir los parámetros correctamente a través del formulario
- La segunda prueba consistía en introducir parámetros correctos, dejando siempre algún campo vacío
- La tercera prueba consistía en rellenar el formulario completo con datos erróneos
- La cuarta prueba, en pos de verificar la seguridad del sistema, se introducía en todos los campos del formulario una prueba básica de SQL Inject que consistía en introducir la cadena de texto a’ and 1=1; --

Los resultados obtenidos, no siempre han sido los esperados, y se ha tenido que retocar la funcionalidad web para adaptarla a lo que se requería de ella, pero tras el descubrimiento de errores, se puede asegurar, que la web funciona correctamente y es segura ante ataques de ***SQL Injection*** básicos. Esto se puede asegurar tan categóricamente, gracias a que, tras las correcciones pertinentes y volver a pasar el banco de pruebas, todas las nuevas funcionalidades de cada *sprint* devolvían lo que debían devolver en cada uno de los casos de prueba. En el primero devolvían el contenido solicitado, ya sea una lista de partidos, un resultado de un partido en concreto, etc. Y en las otras tres pruebas, la funcionalidad devolvía un error controlado indicando que había sido imposible realizar la consulta ante la falta de datos o la naturaleza incorrecta de estos.

Las pruebas unitarias sobre la capa de la **base de datos**, han resultado ser muy similares a las realizadas contra la capa web. En este caso se probaba que las consultas devolviesen los resultados que tenían que devolver, mediante la creación de consultas estándar sobre las que había que ir variando los datos para las distintas pruebas. También se probó que las relaciones entre tablas fuesen correctas, que los índices, las *primary key* y las *foreign key* funcionasen como es debido. Para ello se crearon otras consultas que se iban variando según la prueba. Estas consultas consistían en introducir un dato que no existía en la tabla con la que se relacionaba la tabla sobre la que se introducía. Se probaba a realizar búsquedas en tablas por el campo índice y por campos que no fuesen índices para comprobar la eficiencia, etc. Y, por último, se probaba la inserción de datos en campos críticos, como pueden ser aquellos *NOT NULL*, y dejar el campo vacío, introducir datos repetidos en campos *primary key* o *unique.*

El resultado de todas estas pruebas, al contrario que en la capa web, resultó satisfactorio desde el primer momento, por lo que no hubo que realizar modificaciones de bugs en las nuevas tablas. Por esta razón, se puede asegurar que la base de datos, a lo largo de todos los *sprints* realizados ha sido y es completamente funcional y ha cumplido y cumple con todos los requisitos que se le exigen al *sprint* y al proyecto completo.

Por último, las pruebas realizadas sobre la aplicación Android y sus clases se han realizado de dos maneras distintas. Aquellas que no necesitaban la intervención del usuario, como puede ser solicitar un dato a la capa web y recoger la respuesta, se han probado con test unitarios en la herramienta sobre la que se ha desarrollado toda la capa de la aplicación Android, que ha sido *Android Studio* en la versión, actualmente, 2.1.1

Los test unitarios de la herramienta de desarrollo han sido muy variopintos, desde aquellos que solicitaban una serie de datos a la capa web y simplemente los recogían, hasta aquellos en los que se le pasaba como parámetro una cadena de caracteres, se encargaban de formatearlo, enviarlo a la capa web, recibir la respuesta, y volver a formatearla para más adelante pasársela a la vista.

Para acabar las pruebas de cada *sprint* siempre se realizaba una prueba sobre la aplicación completa, con la idea de probar aquella funcionalidad que no ha podido ser probada en las anteriores pruebas unitarias, como puede ser, que la funcionalidad tras pulsar un botón sea la correcta, o que muestra bien los datos que el controlador le pasa a la vista, etc. Estas pruebas, también han servido como pruebas de integración, pues se comprobaba que la aplicación funcionaba y además lo hacía como tenía que hacerlo.

# 8. Conclusiones y trabajo futuro

En este último capítulo de la memoria se desarrollarán las opiniones personales del autor acerca del proyecto realizado, los conocimientos adquiridos y sobre lo que ha supuesto el trabajo realizado de cara al futuro. Además, se incluirá un apartado en el que se comentaran aquellas futuras mejoras que podrían incluirse en el proyecto de cara a complementar la funcionalidad actual.

Sobre el **proyecto desarrollado** se puede afirmar que se ha tratado de un proyecto complejo y de amplias dimensiones que han abarcado diversos campos dentro de lo que es el software. Se ha tratado de una aplicación Android que consume un servicio RESTful de PHP, el cual obtiene los datos necesarios de una base de datos relacional. El uso del patrón de diseño MVC es otro de los aspectos a destacar dentro del proyecto.

Además del desarrollo e implementación del software se han realizado otras fases pertenecientes al ciclo de vida de un proyecto de software como son el análisis, el diseño y las pruebas. Respecto al ciclo de vida del proyecto, otro tema que hay que destacar es la adaptación del proyecto a la metodología agile *Scrum* apoyándose en *Kanban*.

Otra de las dificultades encontradas y solventadas ha sido el gran cambio existente entre la versión 4.4.2 de Android (utilizada en clase) y la 5.0 (utilizada en el proyecto), ya que no varía únicamente la interfaz, sino que también se modifica la funcionalidad en cada una de las nuevas APIs que Google publica versión tras versión, haciendo muy complicado el trabajo del desarrollador para mantener la aplicación actualizada a la última versión.

Respecto al **trabajo realizado** es innegable que ha supuesto un reto personal muy grande y que se ha resuelto más que satisfactoriamente. El reto lo ha supuesto principalmente el uso de una metodología que desconocía hasta el momento como es la metodología agile con el uso de *Scrum*, la adaptación al uso de *Kanban*, si bien al principio resulta tedioso y hasta liante, acaba siendo de gran ayuda a la hora de finalizar los *sprints* y tener controlado el trabajo realizado y por realizar. Otra dificultad fue la migración de la primera versión del primer *sprint* de la versión de Android 4.4.2 que era la utilizada a la versión 5.0 que ha sido finalmente la elegida para implementar en el proyecto.

Los **conocimientos adquiridos** y las **ventajas de cara al mundo profesional** que este proyecto me ha otorgado ha sido la ampliación de conocimientos de la plataforma Android respecto a los adquiridos a lo largo de la asignatura existente en la carrera, puesto que el desarrollo de aplicaciones móviles está en auge en el mundo empresarial y profesional. Otro de los conocimientos a destacar es el aprendizaje y la puesta en marcha de un proyecto con la metodología *Scrum*. Al igual que el desarrollo de aplicaciones móviles, la metodología agile es muy demandada por las empresas, y permite la ampliación de fronteras respecto a las distintas formas de afrontar un proyecto según la metodología más apropiada.

También destacan entre las materias aprendidas el uso del tablero *Kanban* y la implementación de un **Servicio Web *RESTful*** con ***PHP*, *MySQL* y *JSON***, y el trato profesional con un cliente real, como han sido los usuarios de la aplicación desarrollada.

Debido a la presentación de Google durante los días previos a la entrega de esta memoria, en su evento Google I/O, de la herramienta Firebase, como **futura mejora** ideada por el autor de esta memoria, se plantea una migración a dicha herramienta.

Una de las ventajas que esta herramienta proporciona es la existencia de hosting gratuito dotado de canal seguro, y con posibilidad de añadir más seguridad desde el código, cosa que no muchos hostings gratuitos permiten en la actualidad. A su vez, es preferible mantener todos los servicios que usa la aplicación en un mismo lugar, y esto sería posible gracias a la base de datos en tiempo real que implementa dicha herramienta.

Se migraría también de actual método de notificaciones entre cliente y servidor, es decir, *Google Cloud Messaging* a Firebase Cloud Messaging o ***FCM***.

Gracias a que actualmente esta herramienta pertenece a Google, se podrían aglutinar varias *APIs* que se planean introducir en futuras versiones de la aplicación como puede ser Google *Maps*.

Una de las **desventajas** de esta migración es el cambio de lenguaje del servicio web, que pasaría de ser *PHP* en la actualidad a *node.js* (basado en *JavaScript*) en la herramienta *Firebase*.

Por parte de los usuarios, se plantean varias futuras actualizaciones de la aplicación que incluyan un marcador sobre el que anotar el tanteo del partido, así como un cronometro que lleve el tiempo del mismo. Desde este marcador, debe poderse anotar las estadísticas de cada jugador de manera más o menos sencilla.

Poder anotar las estadísticas es parte de otra de las mejoras propuestas por los usuarios, que es la implementación de un *manager* de baloncesto con los jugadores de la liga. Este *manager* debe seguir la estela de los actuales *managers* online de futbol o baloncesto, en los que, con un presupuesto inicial, el usuario selecciona un equipo sin quedarse en negativo. Estos jugadores, según los partidos reales que disputan reciben una puntuación en consecuencia de sus estadísticas. Con la puntuación de todos los jugadores del equipo se obtiene una puntuación total que sirve para realizar una clasificación entre todos los usuarios cada jornada, según la cual, se otorgara al usuario un premio de dinero ficticio para que mejore su equipo comprando o vendiendo jugadores.

El usuario añadió una última propuesta de mejora o actualización para la aplicación, que consistirá en la creación de una sección para cada equipo, en la que se podría poner una galería, una sección de noticias, un informe de los partidos jugados, etc. En este apartado, la subida de contenido correspondería única y exclusivamente a los miembros del equipo.

Estas actualizaciones, y todas aquellas que se vayan incluyendo a lo largo de la vida útil de la aplicación irán encaminadas a mejorar el feedback recibido por parte de los usuarios durante la realización de las pruebas de integración y funcionalidad complementaria realizadas por el cliente.

# Referencias

[1] Welling, L., & Thomson, L. (2005). Desarrollo web con PHP y MySQL.

[2] Joseph, S., & Jevitha, K. P. (2016). Evaluating the Effectiveness of Conventional Fixes for SQL Injection Vulnerability. In Proceedings of 3rd International Conference on Advanced Computing, *Networking and Informatics* (pp. 417–426).

[3] Asghar, M. Z., & others. (2016). MySQL Database of Web Based attendance management system.

[4] Armendariz Perez, I., & others. (2016). Análisis de los principales sistemas de gestión de bases de datos ante ataques básicos.

[5] González-Garrido, E. (2016). El lenguaje PHP. Programación de aplicaciones Web utilizando PHP.

[6] Patterns, D. (2003). Model-View-Controller. Microsoft Patterns & Practices, http://msdn. microsoft. com/practices/type/Patterns/Enterprise/DesMVC.

[7] Martin, R. C. (2009). Clean code: a handbook of agile software craftsmanship. Pearson Education.

[8] https://www.owasp.org/index.php/Top_10_2013-Top_10 Ultimo acceso: 25 de junio de 2016

# Glosario

*Sprint*

En términos informáticos, un *sprint* es el desarrollo completo y funcional de una parte de una aplicación. El desarrollo completo de una tarea puede considerarse un único *sprint*. Los *sprints* deben ser lo suficientemente ligeros para hacer que el proceso de desarrollo sea ágil, y a la vez lo suficientemente grande para no convertir un desarrollo ligero en uno largo debido a la cantidad de *sprints*. Lo normal es que duren entre 12 y 14 días (aproximadamente dos semanas). En estos *sprints* se realiza el desarrollo completo para dicha tarea, es decir, análisis, diseño, codificación, pruebas, integración (si es necesario) y entrega al cliente.

*Scrum*

Se trata de un marco de trabajo para el desarrollo ágil de proyectos. No se trata de una metodología como tal. Su principio básico es la entrega temprana al cliente de una parte del proyecto con valor para su satisfacción. Estas entregas son iterativas e incrementales aportando nuevas funcionalidades en cada *sprint*.

*Kanban*

Tablero ideado por *Toyota* sobre el cual se fundamenta una de las bases de la metodología *agile*. Viene del japonés (kan = tarjeta, ban = visual) y existen distintas variantes, desde tableros físicos hasta tableros digitales organizados por herramientas informáticas. Sobre él se muestra el estado del desarrollo en el momento actual. Para ello hace falta dividir cada iteración del proyecto en distintas tareas individuales sobre las cual pueda trabajar un único desarrollador. Es una manera muy eficaz de controlar el trabajo y poder tener control sobre él, sobre todo cuando se acercan las fechas de entrega de iteraciones.

Android

Sistema operativo para dispositivos móviles (teléfonos, tabletas, relojes, y hasta coches y gafas) basado en el *kernel* de *Linux* y desarrollado actualmente por Google. Es el sistema operativo sobre el cual se ha desarrollado la aplicación final para el usuario.

MVC

Patrón de diseño formado por tres capas que le dan nombre (Modelo, Vista y Controlador) las cuales, para favorecer la abstracción y reutilización del sistema, se comunican de manera especial. El modelo proporciona los datos al controlador que se encarga de tratarlos y formatearlos para que la vista los muestre.

PHP

Lenguaje web utilizado en el proyecto para construir el servicio web *RESTful*. Corresponde a la capa del controlador dentro del patrón *MVC*. La aplicación *Android* consume el servicio web y envía una petición en formato *JSON*, que es tratada por el servicio web *PHP* y solicita a la base de datos (modelo) los datos que la vista le ha solicitado, y se los devuelve en formato *JSON*.

| | |
|---|---|
| MySQL | Herramienta de gestión de bases de datos relacionales. Representa dentro del patrón *MVC* la capa del modelo. Recibe consultas preparadas desde el servicio web en PHP y devuelve los datos existentes en las tablas. |
| API | Del inglés *Application Programming Interface*, es un conjunto de métodos y procedimientos que se ofrece a otro software para ser utilizado como una capa de abstracción. La API publica una interfaz que mostrara las funciones y los parámetros de estas que pueden ser llamados. La funcionalidad interna es invisible para el programa que consume la API, se trata de una caja negra. |
| Servicio Web | Se podría decir que son *APIs* colgadas en la web para que otro servicio los consuma. Se trata de una tecnología que utiliza distintos protocolos para comunicarse con las aplicaciones que lo consumen y con las aplicaciones que el propio servicio utiliza. A diferencia de las *APIs*, el servicio web no tiene que estar físicamente dentro del proyecto que lo consume, como si ocurre con las *APIs*. |
| Metodología *agile* | Es la metodología implementada a lo largo del proyecto. Se trata de una metodología conformada por una serie de técnicas para la gestión de proyectos surgidos en contraposición a los métodos clásicos de gestión. Se basa en cuatro principios básicos que son **los individuos y su interacción** por encima de los procesos y las herramientas, **el software que funciona** frente a la documentación exhaustiva, **la colaboración con el cliente** por encima de la negociación contractual y **la respuesta al cambio** por encima del seguimiento de un plan. |

# Anexos

## A. Descripción detallada de un sprint de un proyecto de desarrollo de software

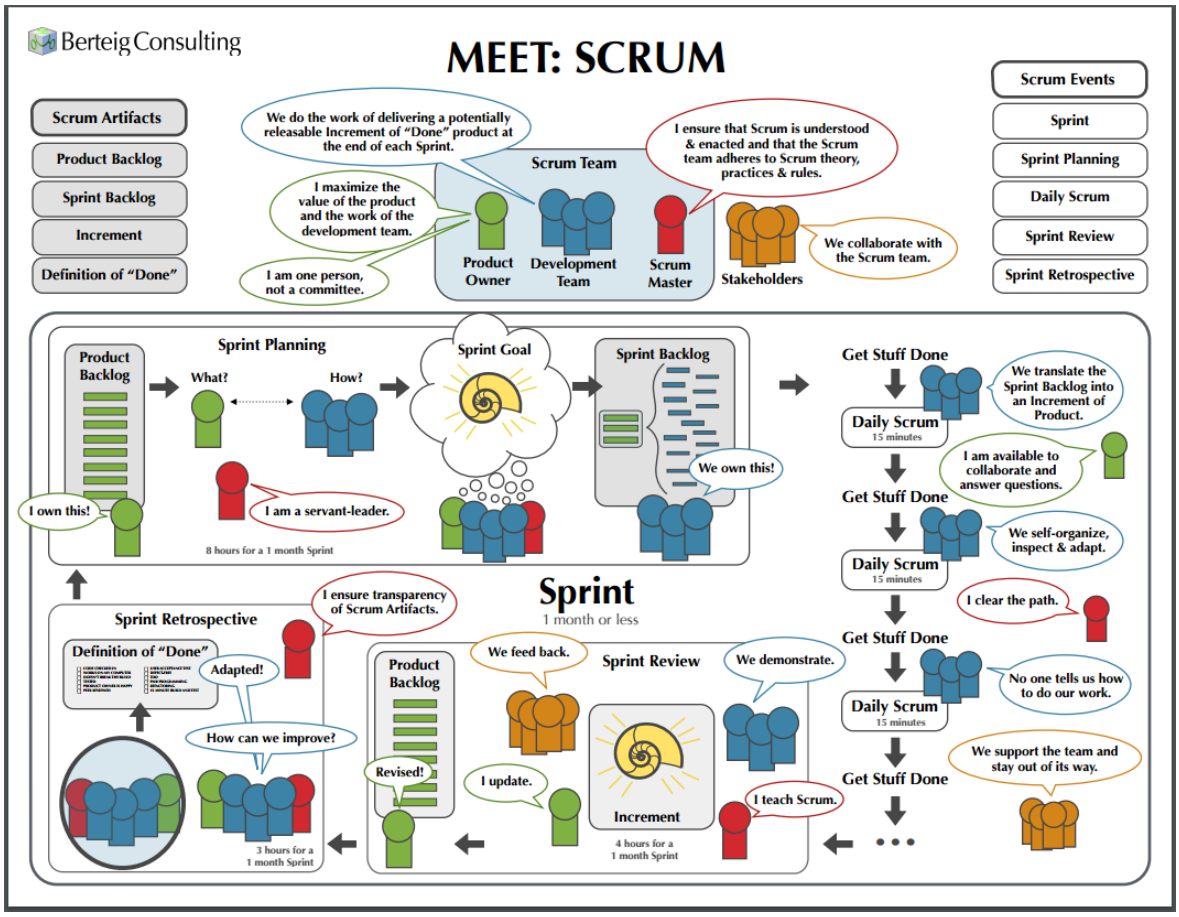



Imagen 18.-Diagrama descriptivo del proceso de *Scrum* para un proyecto. Fuente: http://www.agileadvice.com/2014/03/20/referenceinformation/new-scrum-diagram-meet-scrum-by-travis-birch-csp/

En la Imagen 18 se observa un diagrama del proceso de *Scrum* aplicado sobre cualquier proyecto, y la descripción de sus participantes. Estos son el "propietario del producto" o cliente final, de color verde. De color azul se representa al "equipo de desarrollo", y de color rojo al "*scrum master*". Fuera de lo que sería el *scrum team* se encuentran, de color naranja, lo que se podría definir, como las "partes interesadas" en el avance del producto final.

Un ciclo de vida de un proyecto aplicado a *Scrum*, sería la sucesión de iteraciones o *sprints* de no más de un mes de duración (lo aconsejable es de 2 a 4 semanas) del proceso hasta que el proyecto o el producto está completamente acabado.

Estas iteraciones comienzan con un producto en reserva (*product backlog)*. Acto seguido el propietario del producto (muñeco verde) y el equipo de desarrollo (muñecos azules) comienzan a dialogar sobre lo que quieren y como lo quieren, para acto seguido, entre

todos, incluyendo al *scrum manager* (muñeco rojo) determinar cuál será el objetivo de este *sprint*.

Con este objetivo, el equipo de desarrollo construye lo que se denomina el *scrum backlog*, que no es otra cosa, que una reserva de las pequeñas tareas o *stuffs* pedidas por el propietario del producto para este *sprint*. Esta sería la primera fase, y se la denomina ***sprint planning*** o **planificación del *sprint*.**

La segunda fase del *sprint* consiste en que el equipo de desarrollo vaya desarrollando cada una de las tareas de manera incremental sobre el proyecto. Se deben realizar *daily scrum* o reuniones diarias, de aproximadamente 15 minutos de duración, en las que intervienen todos los agentes relacionados con el desarrollo del proyecto. Es decir, el equipo de desarrollo, encargado de transmitir el avance en la implementación de las tareas, el propietario del producto, como asesor para resolver posibles dudas de funcionalidad que surjan a los desarrolladores, el *scrum master* para mantener el orden y facilitar el camino al equipo.

En la tercera fase se lleva a cabo el ***sprint review*** o **revisión del *sprint*** en la que el equipo de desarrollo muestra los avances con la integración continua, es decir, el avance del producto con entregas funcionales que complementan a las anteriores y el propietario del producto se encarga de comprobar que esa integración continua es correcta y el producto se mantiene funcional y sin errores. Además, este es el punto en el que el propietario del producto revisa el *product backlog* y lo actualiza a nuevas funcionalidades o necesidades.

Como fase final o ***sprint retrospective*** el equipo completo se encarga de decidir cuál es la definición de ***Done***. Esto puede ser, que se obligue a un código comentado, o refactorizado, u otras “obligaciones” que el equipo acuerde entre todos. Tras esta fase, exceptuando el momento en el que el producto no vaya a ser mejorado más, se comienza una nueva iteración o *sprint* empezando de nuevo por la primera fase

## B. Desarrollo de una tarea en el tablero Kanban y significado de las columnas utilizadas

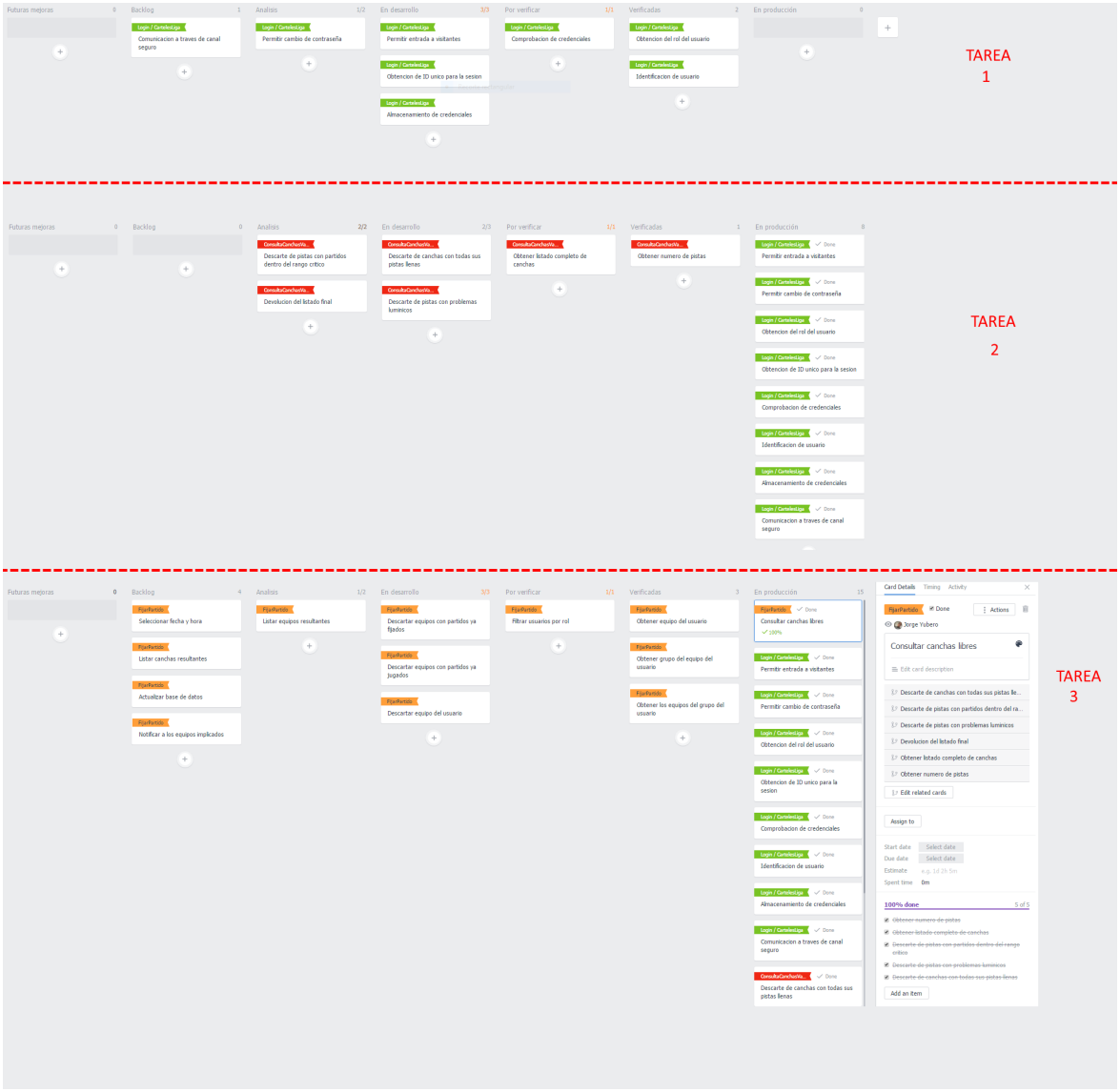



Imagen 19.- Distintos momentos del tablero Kanban a lo largo del desarrollo. Fuente: Propia

En la Imagen 19 se pueden observar tres momentos concretos del tablero Kanban a lo largo de las tres primeras tareas. Se considera que estos tres estados son suficientes para explicar el proceso seguido a lo largo del desarrollo del proceso. Como se ha comentado anteriormente, existen **siete columnas distintas**, las cuales corresponden cada una a una de las tres grandes columnas (esta parte será desarrollada más adelante en sus epígrafes correspondientes).

A lo largo de la **Tarea 1** (color verde), no existe ninguna tarjeta en producción, puesto que se considera que una tarjeta pasa a producción cuando se entrega al cliente, y en el caso de esta aplicación en concreto, hasta que no estuviesen todas las tarjetas en **Revisadas**, la tarea no era entregada al cliente, y, por tanto, no se pasaban a la columna en producción (como si se observa que aparecen en la **tarea 2**).

En la parte de la imagen correspondiente a la **Tarea 2** (color rojo), se ve que ya hay tarjetas en la columna de en producción (todas las de la **tarea 1**), y el resto de tarjetas de la tarea se reparten a lo largo del tablero según su estado (será explicado más adelante a lo largo de esta memoria)

En la **Tarea 3** (color naranja), se observa una columna auxiliar, expandida por el propio programa. En esta pestaña, se observa que una tarea seleccionada (de color naranja, en la columna de producción, arriba a la derecha). Como se comentó, anteriormente, hasta que no estuviesen todas las tarjetas en **Revisadas**, ninguna podría pasar a E**n Producción**, pero en este caso, dicha tarjeta corresponde a la funcionalidad completa desarrollada en la **tarea 2**. Por eso mismo, en la pestaña auxiliar desplegada por el programa, se puede observar que esta tarjeta tiene subtareas, marcadas como completadas y tachadas automáticamente por el programa.

Una vez desarrollada la explicación de los pasos que siguen las distintas tareas de un *sprint* a lo largo del tablero *Kanban*, la memoria se centrará ahora en abordar y explicar el uso otorgado a cada una de las 7 columnas por las que puede pasar una tarjeta dentro del tablero.

- ***Futuras mejoras (ToDo)***

En esta columna se encuentran todas las tarjetas con ideas futuras para mejorar la aplicación. Son ideas que no se incluirán en la primera versión final de la aplicación, sino que serán añadidas de manera gradual en versiones futuras. Como se comenta en apartados posteriores, tras ya tener acabada la parte funcional de la primera versión de la aplicación, se produjo una reunión con parte de la liga (el cliente) para presentar el producto en sociedad y escuchar las críticas de quienes ya habían probado la aplicación anteriormente. En esta reunión se acordaron diversas mejoras para un futuro de la aplicación. Estas mejoras serían las diversas tarjetas de esta columna del tablero.

- ***Backlog (ToDo)***

En esta columna, a día de hoy, no existe ninguna tarjeta, ya que en esta columna se almacenan las tarjetas que faltan por completar de cara a finalizar el *sprint*. Por ejemplo, en el primero de los *sprints*, que consistía en la entrega al cliente un sistema de introducción de marcadores de los diversos partidos que se jugasen, nada más iniciar el desarrollo de esta tarea, todas las tarjetas se añadirían a backlog. En el ejemplo que no atañe, las tarjetas que se añadieron fueron: creación de base de datos (que englobaba la creación de la tabla de equipos y la de partidos), adaptación del entorno de desarrollo (instalando los paquetes necesarios tanto para el desarrollo, como para la comunicación de la aplicación con la web), creación de la web PHP con la funcionalidad de actualizar marcador de partido, en el que recibirá como argumentos el nombre los equipos y el marcador obtenido, para actualizar en base de datos.

- ***Análisis / Diseño (Doing)***

En esta columna se han ido colocando las tarjetas sobre las que había que realizar el proceso de análisis (el cual se desarrollará en apartados posteriores), y el de diseño. Es el primer paso de una tarjeta dentro de este grupo. Tiene un límite de máximo dos tarjetas conviviendo simultáneamente.

- ***En desarrollo (Doing)***

En esta columna, la cual tiene un máximo de tres tarjetas simultáneamente, se colocan aquellas que ya han sido analizadas y diseñadas y están en proceso de codificación.

- ***Por verificar (Doing)***

En esta columna, están las tarjetas cuyas tareas han sido ya codificadas y están a la espera de pasar un banco de pruebas. En un ciclo de vida clásico, correspondería a la fase de pruebas. Tiene un límite de como máximo una única tarjeta conviviendo en esta columna.

- ***Verificadas (Doing)***

En esta columna, se colocan todas aquellas tarjetas que ya han sido probadas, pero que no han sido instauradas en la aplicación final para entregar al cliente. Suelen habitar esta columna aquellas tarjetas que dependen de otras y se encuentran en pasos anteriores. Por ejemplo, para la tarea de *login*, se necesita que la aplicación recoja datos y los envíe a la web para que los contraste con la base de datos. Si la base de datos o la web aun no tuviesen sus tareas completadas, la tarea perteneciente al cliente Android, debería esperar aquí a que las otras tareas estuviesen acabadas antes de llegar al último paso de su ciclo.

- ***En producción (Done)***

En esta columna, como es obvio, se encuentran las tarjetas cuyas tareas ya están analizadas, diseñadas, codificadas y probadas y se encuentran ya integradas en la aplicación final, pueden haber sido ya entregadas o no al cliente, pero en este punto, se tiene total seguridad que la aplicación es completamente funcional.

## C. Principios del manifiesto de desarrollo agile de 2001

El manifiesto, redactado en 2001 por Kent Beck, Mike Beedle, Arie van Benekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Greening, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland y Dave Thomas, declara:

"Estamos descubriendo formas mejores de desarrollar software tanto por nuestra propia experiencia como ayudando a terceros. A través de este trabajo hemos aprendido a valorar:

**Individuos e interacciones** sobre procesos y herramientas

**Software funcionando** sobre documentación extensiva

**Colaboración con el cliente** sobre negociación contractual

**Respuesta ante el cambio** sobre seguir un plan

Esto es, aunque valoramos los elementos de la derecha, valoramos más los de la izquierda."

Y los principios sobre los que se basa el manifiesto (el texto original y entre paréntesis traducido al castellano) son:

- *Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.* (Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor)

- *Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.* (Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos agiles aprovechan el cambio para proporcionar ventaja competitiva al cliente)

- *Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale. (*Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible)

- *Business people and developers must work together daily throughout the project. (*Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto)

- *Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done. (*Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y apoyo que necesitan, y confiarles la ejecución del trabajo)

- *The most efficient and effective method of conveying information to and within a development team is face-to-face conversation. (*El método más efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara)

- *Working software is the primary measure of progress. (*El software funcionando es la medida principal de progreso)

- *Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely. (*Los procesos agiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de manera indefinida)

- *Continuous attention to technical excellence and good design enhances agility. (*La atención continua a la excelencia técnica y al buen diseño mejora la agilidad)

- *Simplicity--the art of maximizing the amount of work not done--is essential. (*La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial)

- *The best architectures, requirements, and designs emerge from self-organizing teams. (*Las mejores arquitecturas, requisitos y diseños, emergen de equipos auto-organizados)

- *At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly. (*A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia)

## D. Diagrama de clases de la aplicación

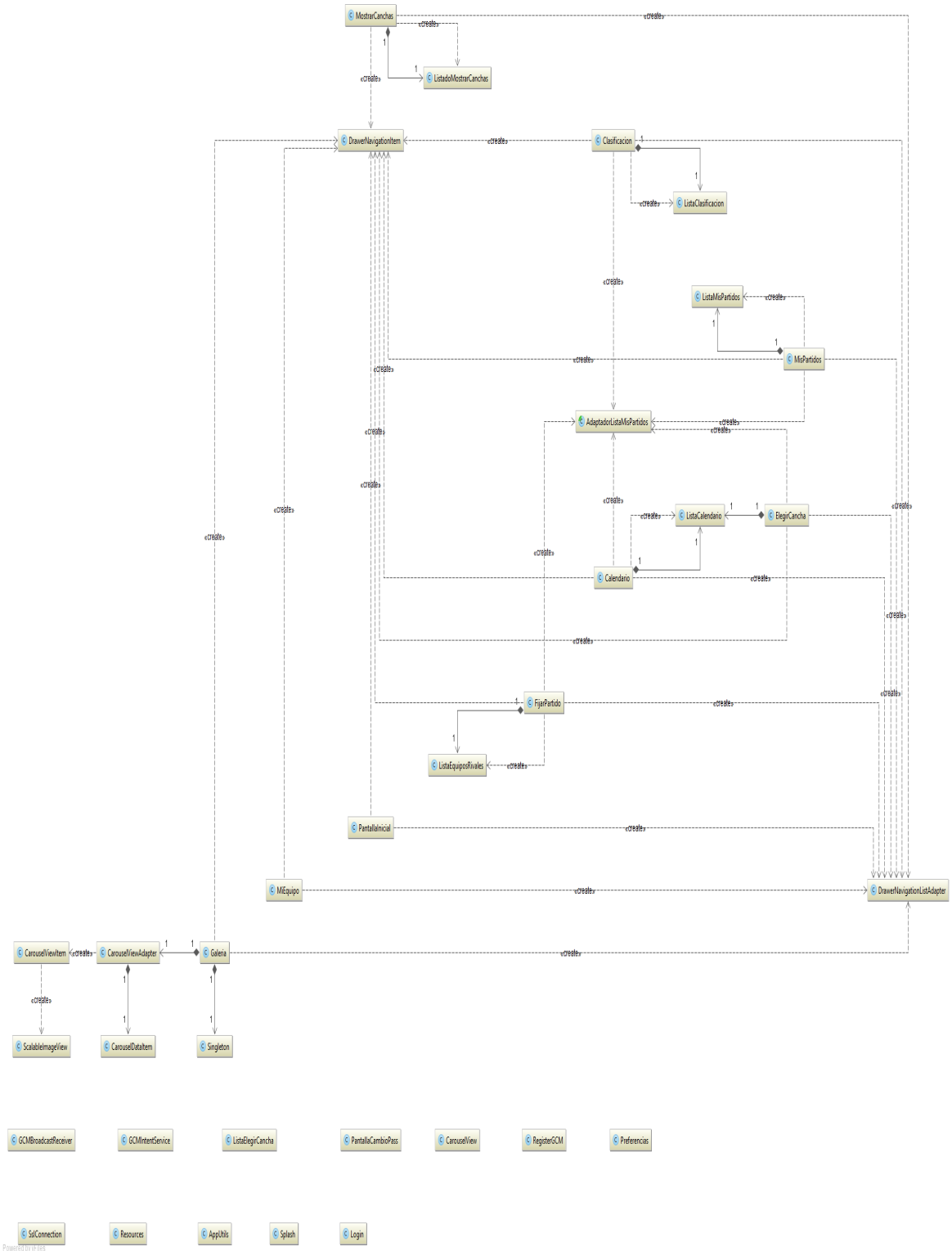



Imagen 20.- Diagrama de clases de la aplicación Android. Fuente: Propia